Universidad Carlos III de Madrid

Escuela Politécnica Superior

Universidad
Carlos III de Madrid

Estudio sobre Android: caso de uso en el mercado empresarial (Droid Preseller)

Proyecto Fin de Carrera

Ingeniería Técnica en Telecomunicaciones en Telemática

Junio 2012

AUTOR:	Raquel Bueso Tamaral
TUTOR:	Vicente Palacios Madrid

A Teito,

por recordarme que era menester hacer el proyecto.

Agradecimientos

Llegado a este punto tengo mucha gente a quien agradecer el haber podido llegar hasta aquí.

A mis amigas Silvi, Ana, Pili, Cris, Débora, Bego, Patri, Jessi, Nuria y Rocío gracias por llevar ahí toda la vida, viviendo buenos momentos junto a mí y lo que es más importante por haber estado tan cerca en los malos. Esta dedicatoria es mi forma de daros las gracias y deciros lo maravilloso que me parece el simple hecho de estar juntas. Y a mis amigos por supuesto gracias por haber estado ahí todos estos años.

A mis amigos de la universidad gracias por hacer que cada vez que pienso en esos años una sonrisa se haga presente. Siendo todos tan diferentes conseguimos fraguar una amistad de la que me siento muy orgullosa y me encanta que sigamos manteniendo. Gracias a todos vosotros por todo lo pasado y lo que esta por venir.

A mis amigos de Leganés gracias por haberme tratado como una más desde el primer día. Por esos maravillosos viajes y tantísimos buenos momentos que pasamos juntos.

A los amigos de Acehuche, gracias por estar siempre ahí y demostrar una y otra vez que la distancias no es un inconveniente para la amistad.

Tengo que agradecer a mi tutor Vicente por todo el apoyo mostrado durante el desarrollo de este proyecto, el haber encontrado un amigo además de un tutor ha sido de gran valor para mí, realmente, gracias por todo.

Tengo que agradecer a Telynet la gran oportunidad que me han dado, y la confianza que han depositado en mí desde el primer momento.

A Paqui, Eugenio, Uge, Maribel, Alberto y Cristina, gracias por estar siempre ahí.

A mi familia, mis tíos, primos y abuelos, haciendo una mención especial a mis abuelas por haber sido tan especiales y a mis tías Pauli, Pepi, Ale y Lali, por quererme tantísimo.

Quiero agradecer a mis padres Teo y Cati, el ánimo y apoyo que me han brindado a lo largo de todos estos años. La confianza que siempre han depositado en mí y el cariño que me han demostrado día tras día. Vuestra forma de inculcarme la necesidad de estudiar sin obligarme jamás a ello creo que ha sido fundamental para llegar hasta aquí.

A Javi y a Carmen gracias por quererme tal y como soy y haberme apoyado siempre en todo. Gracias Javi, por todo lo que tengo que agradecerte en la vida, y puedes estar seguro de que todos mis logros llevan tu firma impresa.

Y por último a Carlos, por hacer que mi mundo gire cada día. Gracias por hacer que los años de la universidad solo me traigan buenos recuerdos y los que han venido después no hayan dejado de mejorar.

Resumen

Hoy en día, los dispositivos móviles y, más concretamente, los teléfonos móviles de última generación, constituyen una realidad que ofrece al usuario no sólo una forma de comunicarse, sino también de divertirse, aprender y, porque no, realizar algunos tipos de trabajos que requieren movilidad y no pueden desarrollarse en un puesto fijo.

En este documento se trata de ofrecer una panorámica general acerca del desarrollo de aplicaciones para dispositivos que dispongan del sistema operativo Android de Google aplicado al mercado de hoy en día. Su principal objetivo es demostrar la considerable mejora que se puede ofrecer a una empresa en su proceso de generación de negocio si deciden utilizar las últimas tecnologías disponibles en telefonía móvil.

El presente proyecto busca conocer y comprender las características y el funcionamiento de este nuevo sistema operativo, evaluando sus posibilidades y ventajas frente a otras alternativas, para ello se estudiarán las características técnicas de la plataforma mencionada analizando las ventajas e inconvenientes respecto a otros entornos de desarrollo disponibles. Además, se incluirá un caso práctico de desarrollo de una aplicación para la plataforma que abarca las fases de análisis, diseño e implementación dentro del proceso de desarrollo del software.

Palabras clave: Android, aplicación, análisis, diseño, implementación.

Abstract

Nowadays, mobile devices and, more concretely, last generation mobile phones, are a reality that offers to users not only a way of communication, but also a way of enjoying, learning and, why not, doing some kind of jobs that require mobility and cannot be done in fixed places.

This document tries to offer a general prospect of developing software for Android OS powered devices and their application in today´s market. It´s main goal is to demonstrate the considerable improvement that a company can achieve in its business generation process if they decide to use the latest mobile technology available.

This project aims to let the reader know and comprehend the characteristics and the behaviour of this new operating system versus other alternatives. Therefore, a complete study of the mentioned platform will be done, analysing pros and cons compared to other development environments available. Besides, a practical case of an application development will be studied, including the analysis and design phases and also the implementation in the software life cycle.

Key words: Android, application, analysis, design, implementation.

Índice general

Lista de Ilustraciones

Lista de Tablas

Lista de Ejemplos

Capítulo I. Introducción

En las siguientes líneas se hace una breve introducción al presente proyecto, exponiendo cuál es su motivación, qué objetivos son los que persigue y cuáles son los contenidos ofrecidos en esta memoria.

1.1 Motivación

Al echar la vista atrás y observar el desarrollo tecnológico que ha experimentado la Humanidad desde mediados del siglo XX hasta hoy, no cabe duda de que más que un avance se ha producido una verdadera revolución. El descubrimiento de la informática, su aplicación paulatina en todo tipo de áreas de conocimiento y de producción, así como su introducción en el común de la población a través de todo tipo de componentes ha cambiado nuestra sociedad y nuestra economía más rápido que cualquier otro hecho o descubrimiento anterior.

En 1983 Motorola comenzó a vender el *DynaTAC 8000X*, el primer teléfono móvil. En ese momento nadie podía imaginar lo rápido que estos dispositivos se expandirían y la importancia que adquirirían en la vida cotidiana de miles de millones de personas a nivel mundial. Hoy en día los teléfonos móviles son parte de la vida de muchas personas y para la mayoría se han convertido en un elemento indispensable que les conecta con el resto del mundo. Sin embargo, los teléfonos móviles han evolucionado tanto que ya no se utilizan únicamente para proporcionar un medio de comunicación entre las personas. Los terminales más avanzados, llamados “teléfonos inteligentes” (del inglés *smartphone*), como el iPhone 4 o los terminales Android pueden ejecutar miles de aplicaciones capaces de llevar acabo casi cualquier tarea imaginable. Estos teléfonos se han convertido en pequeños ordenadores de bolsillo que incluyen las funcionalidades de otros muchos dispositivos, hasta el punto de que los están remplazando. Navegadores GPS, reproductores MP3, cámaras de fotos y video, mandos a distancia, calculadoras y muchos otros aparatos están siendo sustituidos por los teléfonos móviles.

1.2 Objetivos

El nuevo sistema operativo para dispositivos móviles de Google, Android, centra el desarrollo de este proyecto fin de carrera. Para poder dirigir con mayor éxito los esfuerzos por conocer y comprender las características de este nuevo sistema, es necesario fijar unos objetivos que abarquen las actividades que se pretenden realizar y, además, permitan al final de las mismas conocer el grado de desarrollo y cumplimiento alcanzado. Por ello, los objetivos perseguidos en el desarrollo de este proyecto fin de carrera son los enumerados a continuación:

- **Ofrecer una panorámica de las distintas plataformas móviles que existen en el mercado.** Se realizará un estudio de las plataformas móviles que hay en la actualidad analizando las posibilidades que ofrece cada una para un desarrollador de software.

- **Conocer las principales características de Android**. El primer paso para conocer este nuevo sistema debe consistir en indagar toda la información posible sobre él, a fin de conocer cuál es su arquitectura, sus componentes básicos, y cuál es su comportamiento al ejecutar las aplicaciones, documentando todos estos aspectos. Además, ha de averiguarse cuáles son las ventajas y las posibilidades reales que Android ofrece frente a otros sistemas de similar naturaleza.

- **Estudiar el entorno de desarrollo de Android**. Al lanzarse bajo una licencia de software libre, el SDK completo está disponible para cualquier desarrollador que desee descargarlo. Este incluye numerosas ayudas para comenzar a crear aplicaciones en Android, desde las APIs completas con todas las clases y paquetes, hasta herramientas de programación y un completo emulador para poder realizar pruebas. Todos estos elementos han de ser estudiados y explicados.

- **Desarrollar una aplicación completa para Android**. Una vez conocidas las características de este sistema, así como el entorno de desarrollo que ofrece y sus posibilidades, debe crearse una aplicación que aproveche algunas de sus características más fundamentales tomando el sector empresarial como ejemplo del ámbito de la aplicación.

Siendo un poco más concretos, y a modo de resumen, los objetivos son:

- Estudiar las tecnologías móviles del mercado.
- Estudiar el desarrollo de aplicaciones en Android.
- Diseñar la aplicación.

- Implementar la aplicación.
- Probar la funcionalidad de la aplicación.
- Documentar el diseño, implementación y pruebas.

1.3 Estructura del proyecto

A continuación se explica brevemente el contenido de cada capítulo incluido en esta memoria.

En el capítulo actual (*Capítulo I. Introducción*) se expone la motivación del presente proyecto, los objetivos que persigue, así como una visión general de los contenidos de la memoria.

En el *Capítulo II. Estado del arte* se ofrece una descripción general de algunos de los aspectos técnicos relacionados con este proyecto, como los dispositivos móviles, los sistemas operativos más extendidos para estos. Llegando a realizar un estudio más extenso sobre *La Plataforma Android*, se explica al lector las características básicas del nuevo sistema operativo Android, su diseño, arquitectura y funcionamiento.

En el *Capítulo III. Tecnologías y herramientas* se describen las herramientas y tecnologías utilizadas, detallando la instalación de un entorno de desarrollo.

En el *Capítulo IV. Desarrollo del proyecto* se describe de forma detallada el desarrollo de la aplicación Android: desde su análisis y diseño, hasta la implementación de algunas de sus clases y componentes básicos, haciendo énfasis en la usabilidad que se le ha querido dar a la aplicación para un posible caso de uso empresarial. En este capítulo se mostrará un completo manual de usuario de la aplicación así como las pruebas realizadas para cerciorarse de su correcto funcionamiento. También se ofrece al lector una breve explicación de los pormenores en la vida del proyecto, la evolución de su desarrollo en el tiempo y un presupuesto justificado.

En el *Capítulo V. Conclusiones y desarrollos futuros* se exponen los resultados obtenidos tras la finalización del proyecto, comparándolos con los objetivos marcados inicialmente, y trazando posibles líneas futuras de desarrollo.

El *Capítulo VI. Glosario* contiene los términos utilizados a lo largo de proyecto.

El *Capítulo VII. Bibliografía* contiene las referencias bibliográficas empleadas para la realización de la memoria.

En el *Anexo* final se ha incluido la configuración necesaria que se ha tenido que implementar del servidor de aplicaciones necesario para la realización de las pruebas de la aplicación.

Capítulo II. Estado del arte

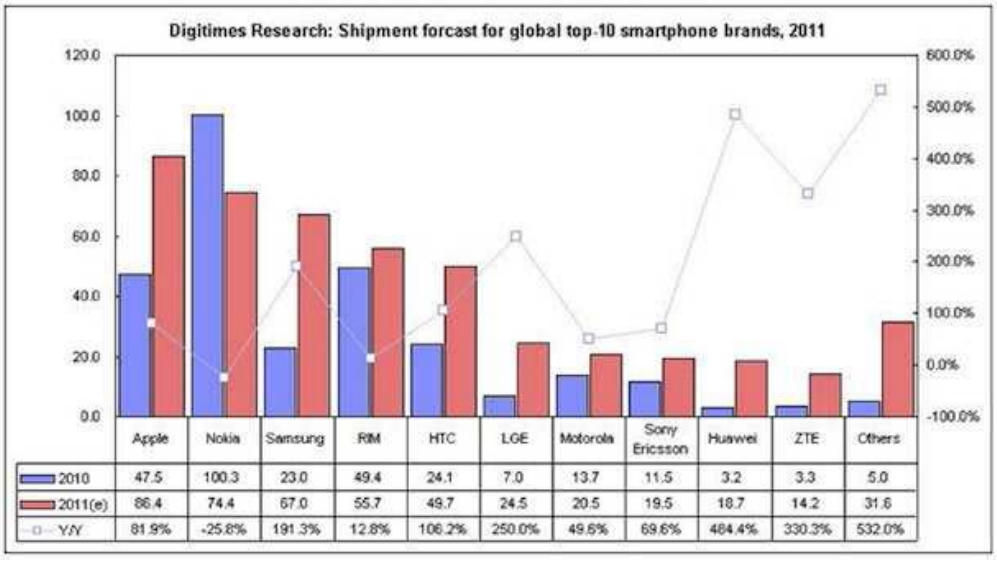
2.1 Tecnologías móviles

Se podría afirmar que la vida empresarial está totalmente vinculada al avance de la tecnología. Las empresas que aprovechan antes las ventajas que ofrece el uso de dispositivos móviles en sus procesos internos, están teniendo una gran ventaja competitiva sobre el resto. (Parrés, 2011)

La tecnología avanza sin parar y nos ofrece grandes oportunidades para agilizar el uso de las aplicaciones corporativas. ¿Y si combinamos el uso de dispositivos móviles con la computación en la nube (*Cloud Computing*)? Tendremos, en un corto espacio de tiempo, soluciones de movilidad total, no sólo para los directivos, sino también de gran utilidad al alcance de ciertos departamentos, y en algunos casos, de utilidad para todos los empleados.

Cuando hablamos de *smartphones* los números son impactantes:

- Crecimientos anuales de hasta un 200%.
- Cada fabricante mide sus ventas en decenas de millones de unidades.
- En cinco años estaremos hablando de un crecimiento del 500%.



	Apple	Nokia	Samsung	RIM	HTC	LGE	Motorola	Sony Ericsson	Huawei	ZTE	Others
2010	47.5	100.3	23.0	49.4	24.1	7.0	13.7	11.5	3.2	3.3	5.0
2011(e)	86.4	74.4	67.0	55.7	49.7	24.5	20.5	19.5	18.7	14.2	31.6
Y/Y	81.9%	-25.8%	191.3%	12.8%	106.2%	250.0%	49.6%	69.6%	484.4%	330.3%	532.0%

Ilustración 1. Evolución de ventas de las diez primeras marcas

Este análisis podría extenderse a las *Tablet PC*. La pregunta es: ¿deben las empresas tener estos datos en cuenta en sus planes estratégicos? Sin duda alguna, la respuesta es afirmativa.

2.1.1 Evolución

Los teléfonos móviles son hoy en día una parte fundamental de la sociedad para bien o para mal, pero incluso ellos tuvieron un origen. En los últimos cuarenta años, los móviles han experimentado un salto evolutivo impresionante. El camino que han recorrido ha sido largo: desde unidades que debían ser cargadas en un bolso hasta los más modernos *smartphones*. (Abc.es, 2011)

A finales de la década de los setenta, la línea que definía a los teléfonos portátiles era un poco borrosa. Algunos eran demasiado grandes y pesados para ser cargados, por lo que se solían colocar en coches, barcos y otros vehículos. El siguiente paso era inevitable. Los teléfonos fueron poco a poco reduciendo su tamaño para permitir su traslado personal. Los primeros ejemplares fueron muy toscos, pero cumplían con su objetivo.

En la década de los ochenta, el concepto de teléfono móvil buscó su punto de maduración. Ya en los noventa, los esfuerzos de muchas empresas se volcaron hacia la reducción de tamaño y a la ampliación de la duración de la batería.

Con la llegada del siglo XXI, los objetivos pasaron a ser la accesibilidad y el alcance masivo. En la actualidad, una gran cantidad de ejemplares cuentan con un poder de procesamiento que no tiene nada que envidiar a todo un ordenador.

Todo ha cambiado en el mundo de la telefonía móvil cuando un tal Martin Cooper se detuvo el martes 3 de abril de 1973, hace casi cuatro décadas en la Sexta Avenida para realizar la que fue la primera llamada por teléfono móvil de la historia.

2.1.2 Dispositivos móviles

Los dispositivos móviles (también conocidos *palmtop* o simplemente *handheld*) son aparatos de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales. (Baz Alonso, Ferreira Artime, Álvarez Rodríguez, & García Baniello, 2009)

Dado el variado número de niveles de funcionalidad asociado con dispositivos móviles, en el 2005, **T38 y DuPont Global Mobility Innovation Team** propusieron los siguientes estándares para la definición de dispositivos móviles:

- **Dispositivo Móvil de Datos Limitados (Limited Data Mobile Device)**: dispositivos que tienen una pantalla pequeña, principalmente basada en pantalla de tipo texto con servicios de datos generalmente limitados a SMS y acceso WAP. Un típico ejemplo de este tipo de dispositivos son los teléfonos móviles en blanco y negro.

- **Dispositivo Móvil de Datos Básicos (Basic Data Mobile Device)**: dispositivos que tienen una pantalla de mediano tamaño, (entre 120 x 120 y 240 x 240 píxeles), menú o navegación basada en iconos por medio de una rueda o cursor, y que ofrecen acceso a e-mails, lista de direcciones, SMS, y un navegador web básico. Un típico ejemplo de este tipo de dispositivos son las BlackBerry y los Teléfonos Inteligentes.

- **Dispositivo Móvil de Datos Mejorados (Enhanced Data Mobile Device)**: dispositivos que tienen pantallas de medianas a grandes (por encima de los 240 x 120 píxeles), navegación de tipo stylus (navegación basada en la utilización de un lápiz táctil), y que ofrecen las mismas características que el "Dispositivo Móvil de Datos Básicos" (Basic Data Mobile Devices) más aplicaciones nativas como aplicaciones de Microsoft Office Mobile (Word, Excel, PowerPoint) y aplicaciones corporativas usuales, en versión móvil, como Sap, portales intranet, etc. Este tipo de dispositivos incluyen los Sistemas Operativos como Windows Mobile 2003 o versión 5, como en las Pocket PC.

PDAs (Personal Digital Assintant)

Un PDA, es una computadora de mano originalmente diseñada como agenda electrónica con un sistema de reconocimiento de escritura. Hoy día (2012) estos dispositivos, pueden realizar muchas de las funciones de una computadora de escritorio pero con la ventaja de ser portátil.

Inicialmente los PDAs incluían aplicaciones estrictamente relacionadas con su función como agenda electrónica, es decir, se reducían a calendario, lista de contactos, bloc de notas y recordatorios. Con el paso de tiempo han ido evolucionando hasta los dispositivos actuales que ofertan un rango mucho más extendido de aplicaciones, como juegos, acceso al correo electrónico o la posibilidad de ver películas, crear documentos, navegar por Internet o reproducir archivos de audio.

Las características del PDA moderno son pantalla sensible al tacto, conexión a una computadora para sincronización, ranura para tarjeta de memoria, y al menos Infrarrojo, Bluetooth o WiFi. La irrupción de Microsoft Windows CE (2000) y Windows Mobile (2003) en el sector los dotó de mayores capacidades multimedia y conectividad. Las PDAs de hoy en día traen multitud de comunicaciones inalámbricas (Bluetooth, WiFi,

IrDA, GPS,) que los hace tremendamente atractivos hasta para cosas tan inverosímiles como su uso para domótica o como navegadores GPS.

Teléfonos móviles

El teléfono móvil es un dispositivo inalámbrico electrónico basado en la tecnología de ondas de radio, que tiene la misma funcionalidad que cualquier teléfono de línea fija. Su principal característica es su portabilidad, ya que la realización de llamadas no es dependiente de ningún terminal fijo y no requiere ningún tipo de cableado para llevar a cabo la conexión a la red telefónica. Aunque su principal función es la comunicación de voz, como el teléfono convencional, su rápido desarrollo ha incorporado funciones adicionales como mensajería instantánea (sms), agenda, juegos, cámara fotográfica, agenda, acceso a Internet, reproducción de video e incluso GPS y reproductor mp3.

La evolución del teléfono móvil ha permitido disminuir su tamaño y peso, desde el Motorola DynaTAC, el primer teléfono móvil en 1983 que pesaba 780 gramos, a los actuales más compactos y con mayores prestaciones de servicio. Además a lo largo de estos años se ha llevado a cabo el desarrollo de baterías más pequeñas y de mayor duración, pantallas más nítidas y de colores, la incorporación de software más amigable.

Inicialmente los teléfonos móviles sólo permitían realizar llamadas de voz y enviar mensajes de texto. Conforme la tecnología fue avanzando se incluyeron nuevas aplicaciones como juegos, alarma, calculadora y acceso WAP (*Wireless Application Protocol*, acceso a Internet mediante páginas web especialmente diseñadas para móviles).

SmartPhones

Un *smartphone* (teléfono inteligente en español) es un dispositivo electrónico que funciona como un teléfono móvil con características similares a las de un ordenador personal. Es un elemento a medio camino entre un teléfono móvil clásico y una PDA ya que permite hacer llamadas y enviar mensajes de texto como un móvil convencional pero además incluye características cercanas a las de un ordenador personal. Una característica importante de casi todos los teléfonos inteligentes es que permiten la instalación de programas para incrementar el procesamiento de datos y la conectividad. Estas aplicaciones pueden ser desarrolladas por el fabricante del dispositivo, por el operador o por un tercero.

Los teléfonos inteligentes se distinguen por muchas características, entre las que destacan las pantallas táctiles, un teclado QWERTY en miniatura, un sistema operativo así como la conectividad a Internet y el acceso al correo electrónico. El completo soporte al correo electrónico parece ser una característica indispensable encontrada en todos los modelos existentes y anunciados desde 2007 en adelante.

Casi todos los teléfonos inteligentes también permiten al usuario instalar programas adicionales, normalmente inclusive desde terceros, pero algunos vendedores gustan de tildar a sus teléfonos como inteligentes aun cuando no tienen esa característica. Un claro ejemplo de teléfonos inteligentes son los Motorolas de la serie MOTO Q, Nokia serie E, Nokia serie N, y los más reconocidos por estas características BlackBerry, iPhone o Android. Entre las características más importantes están el acceso a Internet y al correo electrónico, a los programas de agenda, las cámaras integradas, administración de contactos, acelerómetros y algunos programas de navegación así como ocasionalmente la habilidad de leer documentos de negocios en variedad de formatos como PDF y Microsoft Office.

2.1.3 Sistemas operativos

Partiendo de la definición de sistema operativo (Linux Para Todos, 2012): Capa compleja entre el hardware y el usuario, concebible también como una máquina virtual, que facilita al usuario o al programador las herramientas e interfaces adecuadas para realizar sus tareas informáticas, abstrayéndole de los complicados procesos necesarios para llevarlas a cabo.

Podemos deducir que el uso de uno u otro S.O determinarán las capacidades de los dispositivos, y la forma de éstas de interactuar con el usuario. Existen multitud de opciones, si bien las más extendidas son Symbian, BlackBerry OS, Windows Mobile, iPhone OS y Android, además por supuesto de los dispositivos con sistema operativo Linux.

2.1.3.1 Palm OS

Convertido en el auténtico líder del mercado desde su aparición en 1996, comenzó a ceder protagonismo con la llegada del nuevo siglo, hasta que en 2003 el fabricante se vio en la necesidad de dividirse y la plataforma pasó a formar parte de una nueva empresa denominada PalmSource, que en 2005 fue adquirida por la japonesa Access. (Wikipedia, Palm OS, 2012)

Los motivos de este retroceso en cuanto a ingresos de la compañía son fáciles de imaginar, la lenta pero imparable penetración de Windows Mobile, la evolución de decenas de modelos de teléfonos móviles inteligentes (la mayoría con Symbian) y la aparición de la plataforma y los dispositivos BlackBerry comenzaron a dejar en un segundo plano a las PDA que no incorporaban telefonía móvil.

En 2009 se cambio el nombre de Palm OS a webOS y actualmente es propiedad de la empresa HP (Hewlett-Packard). El día 10 de Diciembre de 2011, HP anunció que

seguirá dando soporte al sistema operativo, aunque desde ahora será software libre. HP desea que su futuro desarrollo sea transparente y no exista fragmentación.

2.1.3.2 Symbian

Symbian OS es un sistema operativo para dispositivos móviles y *smartphones*, con librerías asociadas, interfaz de usuario originalmente desarrollado por Symbian Ltd. Es un descendiente del EPOC de Psion (empresa pionera en la fabricación de dispositivos móviles) (Charte, 2001) y es ejecutado exclusivamente en procesadores ARM. En 2008, la compañía Symbian Software Limited fue adquirida por Nokia y una nueva e independiente organización no lucrativa llamada Symbian Foundation se estableció. Esta fundación tuvo como objetivo el crear la plataforma Symbian de código abierto. La plataforma ha sido designada como la sucesora de Symbian OS después del lanzamiento oficial de la fundación Symbian en Abril de 2009. La plataforma Symbian se convirtió en código abierto oficialmente en Febrero de 2010. Los dispositivos basados en Symbian OS suponen un 46.9% del total de ventas de *smartphones*.

Entre las principales características de Symbian se encuentran la multitarea y la protección de memoria como cualquier otro sistema operativo (especialmente los diseñados para ordenadores de sobremesa).
Symbian OS fue creado en base a 3 principios del sistema:

- La integridad y seguridad de los datos del usuario es primordial.
- El tiempo del usuario no debe ser malgastado.
- Todos los recursos son escasos.

Para seguir estos principios de la mejor manera posible, Symbian utiliza un *microkernel*, tiene un enfoque de peticiones y de devolución de llamada a los servicios y mantiene una separación entre la interfaz de usuario y el motor. El sistema operativo está optimizado para dispositivos con poca potencia de batería y sistemas basados en memoria ROM. Las aplicaciones y el S.O. (sistema operativo) mismo siguen un diseño Modelo - Vista - Controlador (MVC).

Posteriores revisiones del S.O. diluyeron este enfoque en respuesta a las peticiones del mercado, más notablemente con la introducción de un kernel en tiempo real y un modelo de seguridad en las versiones 8 y 9. Hay un gran énfasis en conservar recursos que está ejemplificado por los lenguajes de programación específicos de Symbian como *descriptors* y *cleanup stack*.

Existen técnicas similares para conservar espacio en disco (aunque los discos en los dispositivos Symbian son generalmente de memoria flash). Por otra parte, toda la programación en Symbian está basada en eventos y la CPU entra en modo de baja potencia cuando las aplicaciones no están tratando directamente con un evento. Esto se logra a través de un lenguaje de programación llamado objetos activos (*active objects*).

Así mismo, el enfoque de Symbian para los hilos y los procesos está orientado a la reducción de los gastos generales.

El kernel de Symbian (EKA2) ofrece respuesta a tiempo real suficientemente rápida como para construir un teléfono alrededor de él, esto es un teléfono con un único núcleo que ejecute tanto las aplicaciones de usuario como la señalización de la pila. Esto ha permitido a los teléfonos Symbian convertirse en más pequeños, baratos y más eficientes energéticamente que sus predecesores.

El modelo de sistema de Symbian contiene las siguientes capas, desde la más alta a la más baja:

- La capa de la interfaz de usuario.
- Capa de servicios de Aplicación.
  - Java ME.
- Capa de servicios del Sistema Operativo.
  - Servicios genéricos.
  - Servicios de comunicación.
  - Servicios multimedia y gráficos.
  - Servicios de conectividad.
- Capa de servicios básicos.
- Servicios e interfaz de hardware del Kernel.

La capa de servicios básicos es la de nivel más bajo que puede ser alcanzada por operaciones del lado del usuario; esto incluye el servidor de ficheros y la librería de usuario, almacenamiento, repositorio central, base de datos y servicios criptográficos.

En Symbian, una mínima porción del sistema tiene privilegios de kernel; el resto se ejecuta con privilegios de usuario en modo de servidores, de forma que los procesos en ejecución y sus prioridades son manejados por este *microkernel*. Cada una de las aplicaciones corre en su propio proceso y tiene acceso únicamente a una exclusiva zona de memoria.

Desarrollar aplicaciones para Symbian es relativamente sencillo. No es necesario aprender ningún lenguaje de programación nuevo porque permite utilizar lenguajes habituales como Java, C++, Visual Basic o Perl, entre otros, para desarrollar aplicaciones. Este hecho ha permitido que actualmente sean cientos de miles las aplicaciones y utilidades disponibles para Symbian.

Ediciones de Symbian

Symbian contempla cinco tipos de ediciones o series del sistema operativo según las características del dispositivo móvil. La principal diferencia entre ediciones no radica tanto en el núcleo del sistema operativo como en la interfaz gráfica utilizada:

- **Serie60**. El más popular de todos, debido fundamentalmente a que el gigante **Nokia**, uno de los fabricantes más importantes del mundo, ha hecho de Symbian y de su versión Serie60 el núcleo de casi todos sus modelos de *smartphones*. Los dispositivos con Serie60 tiene una pantalla pequeña y un teclado del tipo 0-9#. También lo utilizan fabricantes como Siemens, Samsung y Panasonic.

- **Serie80**. Esta edición, también usada por Nokia, está más orientada a dispositivos que tienen pantalla táctil y permiten multitarea, pudiendo tener varias aplicaciones abiertas simultáneamente.

- **Serie90**. Muy similar a la edición Serie80, sólo que estos dispositivos tienen una pantalla más grande y llevan incorporados sensores táctiles más desarrollados. Utilizan teclados virtuales, reconocimiento de trazos o teclados acoplables mediante, por ejemplo, Bluetooth.

- **UIQ**. La interfaz de esta edición de Symbian se encuentra muy influenciada por Palm OS. Implementan una especie de multitarea virtual, dando al usuario la falsa sensación de poder realizar varias acciones simultáneas; suelen tener un alto coste computacional e influyen negativamente en el tiempo de respuesta apreciado por el usuario. Es utilizado en algunos modelos de Sony Ericcson y Motorola.

- **MOAP**. Esta edición se da únicamente en Japón, principalmente en el fabricante FOMA.

2.1.3.3 BlackBerry

BlackBerry es una línea de dispositivos móviles y teléfonos inteligentes diseñados y desarrollados por la compañía canadiense **RIM** (Research In Motion) desde el año 1996.

BlackBerry funciona como PDA, con libro de direcciones, calendario y listas de tareas. También funciona como reproductor multimedia portátil, con soporte para reproducción de música, video y fotos. BlackBerry es principalmente conocida por su capacidad para recibir e-mails (PUSH) sin importar el servicio de red móvil que esté presente, o a través de una conexión WIFI. BlackBerry es principalmente un teléfono de mensajería y dispone del mayor número de características para este cometido en un *smartphone* hoy en día. Dichas características incluyen autocompletado, autocorrección, soporte para múltiples idiomas, accesos directos de teclado, emoticonos, email (push), facebook (push), notificaciones de MySpace, EBay, mensajería instantánea y un largo etc. Todas estas conversaciones y notificaciones de aplicaciones son mostradas en una aplicación de mensajería unificada, a la cuál otras compañías desarrolladoras pueden acceder también. En otros dispositivos, en cambio, muchas de estas aplicaciones

nombradas tendrían que ejecutarse en segundo plano. El protocolo *PUSH* de BlackBerry da a los dispositivos su propio tiempo de vida de batería. Los datos son comprimidos a través del Servicio de Internet BlackBerry (BIS).

BlackBerry tiene aproximadamente dos tercios menos de transferencia de datos que cualquier otro *smartphone* para suministrar la misma información. BlackBerry tiene alrededor del 18% de las ventas de *smartphones* en todo el mundo, lo que le supone ser el segundo dispositivo más vendido después de Symbian. El servicio BIS de BlackBerry está disponible en 91 países a través de más de 500 operadores de servicios móviles mediante diversas tecnologías móviles.

El primer dispositivo BlackBerry fue presentado en 1999. En 2002, el primer *smartphone* BlackBerry fue puesto a la venta, con soporte para correo PUSH, teléfono móvil, mensajes de texto, servicio de fax por internet, navegación web y otros servicios inalámbricos. Es un ejemplo de dispositivo convergente. El primer dispositivo BlackBerry avanzó en el mercado concentrándose en sus capacidades para e-mail. RIM ofrece actualmente servicio de e-mail a dispositivos no BlackBerry, como el Palm Treo, a través del software BlackBerry Connect. El dispositivo BlackBerry original tenía una pantalla monocroma, pero todos los dispositivos actuales tienen pantallas a color. Todos los modelos excepto los de las series Storm, tienen un teclado integrado QWERTY, optimizado para dispositivos móviles.

BlackBerry OS es un sistema operativo para móviles desarrollado por la empresa canadiense Research in Motion para sus *smartphones* llamados BlackBerry. El S.O. (sistema operativo) ofrece multitarea y soporta dispositivos de entrada especializados tales como trackball, trackpad y pantallas táctiles, que han sido adoptados por RIM para usarlos en sus dispositivos. La plataforma BlackBerry quizá es mas conocida por su suporte nativo al e-mail corporativo a través de MIDP1.0 y, más recientemente, MIDP2.0, lo que permite a través de una red wireless la activación y sincronización con servidores de correo Microsoft Exchange, Lotus Dominio o Novell GroupWise, calendarios, tareas, notas y contactos cuando se use en conjunción con BlackBery Enterprise Server. El SO también soporta WAP 1.2 Actualizaciones del SO pueden estar disponibles de forma automática en los operadores inalámbricos que soporten el servicio BlackBerry OTASL (*over the air software loading*).

Los desarrolladores externos pueden escribir software utilizando para ello el API de BlackBerry aunque las aplicaciones que hagan uso de cierta funcionalidad deberán estar firmadas digitalmente.

El S.O. BlackBerry está orientado a su uso profesional como gestor de correo electrónico y agenda. Desde la versión actual, la cuarta, se puede sincronizar el dispositivo con el correo electrónico, el calendario, tareas, notas y contactos de Microsoft Exchange Server además es compatible también con Lotus Notes y Novell

GroupWise. BlackBerry Enterprise Server (BES) proporciona el acceso y organización del email a grandes compañías identificando a cada usuario con un único BlackBerry PIN. Los usuarios más pequeños cuentan con el software BlackBerry Internet Service, programa más sencillo que proporciona acceso a Internet y a correo POP3 / IMAP / Outlook Web Access sin tener que usar BES. Al igual que en el SO Symbian desarrolladores independientes también pueden crear programas para BlackBerry pero en el caso de querer tener acceso a ciertas funcionalidades restringidas necesitan ser firmados digitalmente para poder ser asociados a una cuenta de desarrollador de RIM.

BlackBerry 6

BlackBerry 6 fue presentado en el WES 2010 (*Wireless Enterprise Symposium*) junto con un video promocional donde se muestra algunas novedades. RIM apuesta que su BlackBerry 6 esta enfocado en el mercado corporativo y no-corporativo (Wikipedia, BlackBerry OS, 2012). La mejor experiencia de este sistema se encuentra en los equipos *touchscreen* (pantalla táctil), aunque RIM asegura que en los equipos que cuenten con un *touchPad* o *trackPad* podrán ejecutarlo ya que ejerce casi la misma función.

Algunas de las principales mejoras fueron: un navegador con tecnología WebKit, experiencia con las redes sociales (facebook, twitter, myspace) y mensajería instantánea (BlackBerry Messenger, Windows Live), posibilidad de ejecutar juegos 3D soporte para escaneo de códigos de barras en 1D/2D, mejoras en el Auto-Focus (Los lentes de la cámara y el Auto-Focus funcionan por separado) y reconocimiento de rostro en la cámara

BlackBerry 7

BlackBerry presentó su nuevo sistema operativo BlackBerry 7 en Mayo del 2011. En un principio esta versión iba a ser numerada como la 6.1, pero la cantidad de nuevas funcionalidades que trajo consigo animó a la compañía a darle el empujón para que pasase de una actualización a ser una versión nueva.

Como novedades de esta versión BlackBerry ofreció una interfaz más sencilla, búsquedas por voz y un navegador compatible con vídeos en HTML5 que cuenta con un motor JavaScript más rápido. Se hizo hincapié en la funcionalidad *Balance*, que divide las aplicaciones y la información del dispositivo para crear una sección de trabajo, otra personal. BlackBerry 7 no está disponible para modelos anteriores, de modo que los terminales con BlackBerry OS 6 no tienen una solución (al menos oficial) para poder instalar la versión 7.

2.1.3.4 Windows Mobile

Windows Mobile es un sistema operativo desarrollado por Microsoft para su uso en *smartphones* y dispositivos móviles. La versión actual es Windows Phone 6.5. Está basada en el kernel de Windows CE 5.2 y cuenta con una serie de aplicaciones básicas desarrolladas con el API de Microsoft. Está diseñado para ser similar a las versiones de escritorio de Windows, tanto estética como funcionalmente. Existe además software desarrollado por terceros y que están disponibles a través del *Windows Marketplace*.

Windows Mobile para Pocket PC tiene estas características en la mayoría de sus versiones:

- **Office Mobile**. Paquete Office adaptado a la versión móvil del sistema operativo.
- **Outlook Mobile**, como gestor de correo.
- **Internet Explorer Mobile**. Navegador web desarrollado por Microsoft para PocketPC que viene por defecto con Windows Mobile y Windows CE.
- **Windows Media Player** para Windows Mobile.
- **Compartición de la conexión a Internet**. Permite a ordenadores asociados compartir sus conexiones internet mediante USB o Bluetooth.
- **Sistema de ficheros coherente**, similar al de las versiones 9x/NT de Windows.
- **Multitarea.**

Hardware

Existen tres versiones hardware de Windows Mobile para distintos dispositivos:

- **Windows Mobile Professional**, que es el que utilizan normalmente los *smartphones* con pantalla táctil
- **Windows Mobile Standard**, que lo utilizan los teléfonos móviles con pantalla normal
- **Windows Mobile Classic**, que es el utilizado en Pocket Pc

Dispositivo Windows Mobile clásico. Pocket PC

Un dispositivo Windows Mobile clásico es básicamente, una PDA usando Windows Mobile que no tiene las funcionalidades de un teléfono. Antiguamente se conocía estos dispositivos como Pocket PC. Fue la plataforma para la que inicialmente se creo el sistema operativo Windows Mobile. Actualmente, dentro de la denominación Pocket PC se incluyen tanto los dispositivos sin funcionalidades de telefonía como aquellos que sí las tienen.

SmartPhones

Windows Mobile es el término que emplea Microsoft para referirse a los *smartphones* que hacen uso del sistema operativo Windows Mobile. Aunque en el amplio sentido de la palabra *smartphones*, tanto los teléfonos Pocket PC como los *smartphones* de Microsoft entran dentro de esta categoría, el lector debe notar que Microsoft utiliza el termino *smartphones* solamente para un hardware más específico que difiere de los teléfonos Pocket PC. Tales *smartphones*, fueron originalmente diseñados sin pantallas táctiles con el objetivo de ser manejados eficientemente con una sola mano y típicamente tenían una resolución de pantalla más baja que la de los PocketPCs. El objetivo de Microsoft con los *smartphones* era crear un dispositivo que funcionara bien como teléfono y como dispositivo de datos.

Windows 7

Microsoft había planeado originalmente continuar con la línea de sistemas operativos Windows Mobile hasta la versión 7 (Windows Mobile 7), basadas todas ellas en actualizaciones de la versión anterior. Sin embargo, Microsoft decidió sustituir este sistema operativo por uno nuevo, llamado Windows Phone 7 Series y que finalmente fue renombrado como Windows Phone 7.

La intención de Microsoft era lanzar este sistema operativo durante el año 2009, pero diversos retrasos y la salida del proyecto de Photon (nombre de una de las actualizaciones de Windows Mobile) obligaron a Microsoft a lanzar una versión mejorada de Windows Mobile 6, Windows Mobile 6.5. Durante el congreso mundial de telefonía móvil en Barcelona en el año 2010, Microsoft mostró detalles sobre Windows 7 tales como la integración con los servicio Xbox live y Zune. Los teléfonos que actualmente ejecutan un Windows Mobile 6.5 no se podrán actualizar a Windows Phone 7.

2.1.3.5 IOS

IOS (anteriormente conocido como iPhone OS) es el sistema operativo para móviles de Apple. Desarrollado originalmente para el iPhone, ha sido utilizado desde entonces en el iPod Touch, iPad y Apple TV. Apple no permite que el sistema operativo sea utilizado en dispositivos desarrollados por otras empresas. A día 1 de Septiembre de 2010, la tienda de aplicaciones de Apple, Apple Store, contiene más de 250.000 aplicaciones para el iOS, que han sido descargadas más de 6.500 millones de veces.

La interfaz de usuario de iOS está basada en el concepto de la manipulación directa de datos, usando gestos táctiles multitáctiles. Los elementos utilizados para la interfaz de usuario son botones, campos de texto, checkboxes, etc. La respuesta de la interfaz a los gestos del usuario es inmediata, lo que proporciona una navegación fluida por la interfaz del sistema operativo. La interacción con el sistema operativo incluye gestos como deslices, toques, pellizcos, los cuales tienen definiciones diferentes dependiendo

del contexto de la interfaz. Se utilizan acelerómetros internos para hacer que algunas aplicaciones respondan a sacudir el dispositivo o al cambiar de modo vertical al apaisado o horizontal.

IOS es un sistema operativo que hereda de Mac OS X, con el cuál comparte la base de Darwin, que lo convierte en un sistema operativo Unix. En iOS hay cuatro capas de abstracción: la capa del núcleo del sistema operativo, la capa de “Servicios Principales“, la capa de "Medios" y la capa de "Cocoa Touch". La versión actual del sistema operativo (iOS 5.1.1 en Mayo de 2012) ocupa más o menos 770 megabytes, variando por modelo.

El sistema operativo fue revelado junto con el iPhone en la conferencia Macworld Conference & Expo el 9 de Enero de 2007 y fue lanzado en Junio de ese mismo año. En un principio, Apple no reveló el nombre del sistema operativo simplemente diciendo que ejecutaba un OS X. Inicialmente, las aplicaciones de terceros no fueron soportadas. Steve Jobs mantenía que los desarrolladores podrían desarrollar aplicaciones web que se comportasen como aplicaciones nativas del sistema operativo. El 17 de Octubre de 2007, Apple anunció que un SDK nativo esta siendo desarrollado y que estaba planeado entregárselo a los desarrolladores en Febrero del siguiente año. En Marzo de 2008, Apple lanzó la primera beta, junto con un nombre para el sistema operativo: iPhone OS.

Las fuertes ventas del teléfono de Apple levantó el interés por el SDK. El Septiembre de 2007, Apple había lanzado el iPod Touch, que tenía la mayor parte de características del iPhone (excepto aquellas de telefonía). Apple también vendió más de un millón de iPhones durante el periodo vacacional de 2007. El 27 de Enero de 2010, Apple anuncia el iPad, con una mayor pantalla que el iPhone o el iPod Touch y diseñado para la navegación por internet, reproducción multimedia y lectura de iBooks.

Durante la presentación del iPhone 4 en junio del 2010, Steve Jobs anunció que iPhone OS pasaría a ser llamado oficialmente como iOS.

2.1.3.6 Android

Fue desarrollado inicialmente por Android Inc., una firma comprada por Google en 2005. El sistema busca, nuevamente, un modelo estandarizado de programación que simplifique las labores de creación de aplicaciones móviles y normalice las herramientas en el campo de la telefonía móvil. Al igual que ocurriera con Symbian, lo que se busca es que los programadores sólo tengan que desarrollar sus creaciones una única vez y así ésta sea compatible con diferentes terminales.

A lo largo de este proyecto se explicará en detalle el funcionamiento de este sistema operativo, ya que es el que se ha utilizado para el desarrollo del mismo.

2.2 Selección de la plataforma

A la hora de desarrollar el proyecto se tuvo que seleccionar una de las plataformas de teléfonos móviles disponibles en el mercado que se ajustase a las necesidades que se tenían tanto de hardware como de software.

En un primer momento debido a su popularidad se pensó en Windows Phone, iOS o Android como plataformas para desarrollar la aplicación móvil. Todas ofrecen una serie de herramientas bastante avanzadas para el desarrollo de las aplicaciones y, además, llevan suficiente tiempo en el mercado como para poder obtener *feedback* en la red en el caso de que surjan problemas, aunque en este caso Windows Phone era la que menos ventajas podía ofrecer llegados a este punto.

Por otra parte, iOS y Android son dos plataformas que permiten desarrollar aplicaciones muy atractivas para el usuario puesto que se instalan sobre dispositivos con interfaz táctil y unas capacidades hardware bastante avanzadas, con procesadores que pueden alcanzar 1GHz e incluso algunos con doble núcleo. En ese sentido Android tiene la ventaja de ser el sistema operativo de muchos terminales de marcas como HTC, Samsung o LG mientras que iOS tan sólo se puede utilizar en dispositivos de Apple. Aunque se puede desarrollar en iOS de una forma mucho más homogénea ya que no hace falta tener en cuenta diferentes tamaños de pantalla, resoluciones, disposición de los botones, etc.

Sin embargo Android ofrece dos grandes ventajas que influyeron en la decisión final de la elección, por un lado se trata de una plataforma bajo la licencia Apache, una licencia libre y de código abierto y por otro lado se trata de la plataforma con mayor cuota actual de mercado con más de un 50% de las ventas realizadas en los últimos meses. Esta alta cuota de mercado es en parte debida a la gran aceptación que ha tenido el sistema operativo entre los distintos fabricantes, como consecuencia el mercado ofrece una gran variedad de dispositivos. (Gartner, 2012)

Operating System	4Q11 Units	4Q11 Market Share (%)	4Q10 Units	4Q10 Market Share (%)
Android	75,906.1	50.9	30,801.2	30.5
iOS	35,456.0	23.8	16,011.1	15.8
Symbian	17,458.4	11.7	32,642.1	32.3
Research In Motion	13,184.5	8.8	14,762.0	14.6
Bada	3,111.3	2.1	2,026.8	2.0
Microsoft	2,759.0	1.9	3,419.3	3.4
Others	1,166.5	0.8	1,487.9	1.5
Total	**149,041.8**	**100.0**	**101,150.3**	**100.0**

Ilustración 2. Relación de ventas mundiales del último trimestre de 2011

2.3 Android

Android es una solución completa de software de código libre para teléfonos y dispositivos móviles. Es un paquete que engloba un sistema operativo, un entorno de ejecución de ejecución basado en Java, un conjunto de librerías de bajo y medio nivel y un conjunto inicial de aplicaciones destinadas al usuario final. Se distribuye bajo una licencia Apache, versión 2, una licencia libre permisiva que permite la integración con soluciones de código propietario. (Android Developers, What is Android?, 2012)

Android ha causado un gran impacto en la industria de la comunicación móvil, estableciendo una plataforma abierta que permita un acceso fácil a prácticamente todas las funcionalidades hardware de los dispositivos en los que esté instalado, así como proveyendo de serie a los desarrolladores con librerías que favorezcan la creación ágil y rápida de aplicaciones. Se ha hecho especial énfasis en que las aplicaciones creadas por terceros no tendrán ningún tipo de desventaja en cuanto a funcionalidad y acceso al dispositivo que las aplicaciones "nativas" que se distribuirán originalmente con Android.

2.3.1 Arquitectura

Android proporciona un paquete completo de software a todos los niveles:

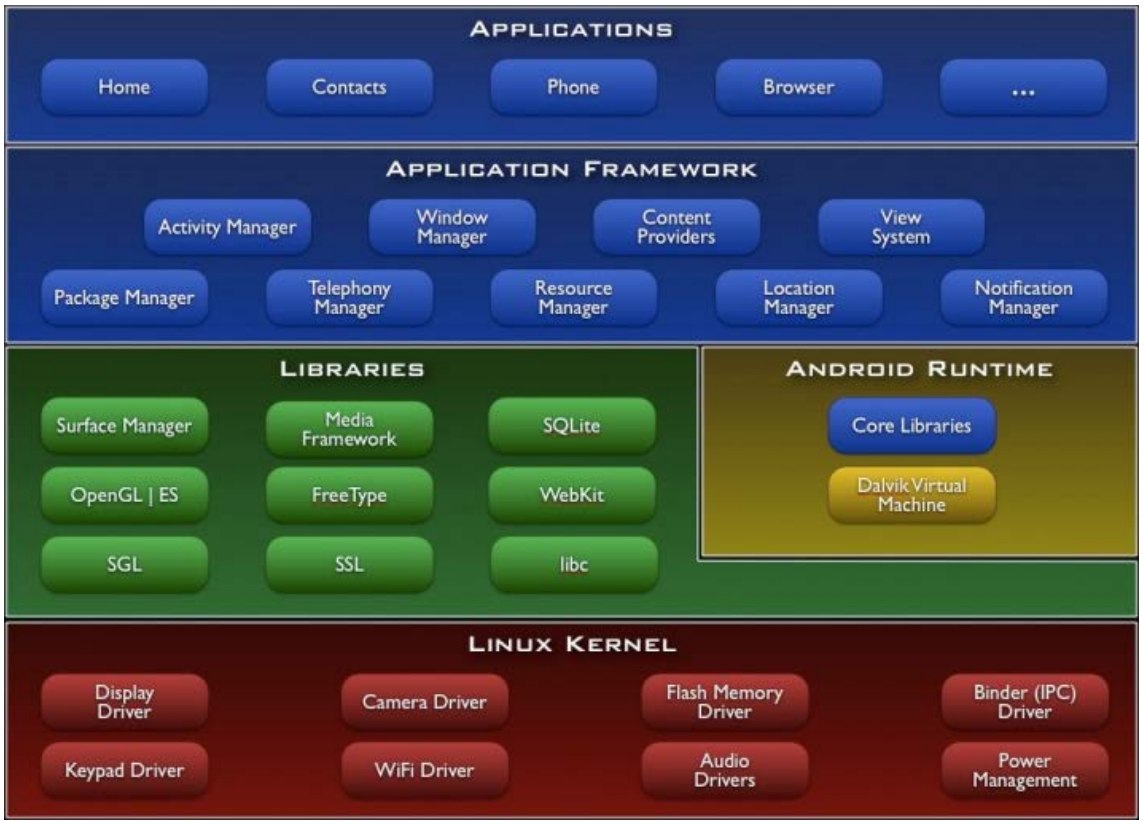


Ilustración 3. Arquitectura de Android

- **Aplicaciones**: las aplicaciones base incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas las aplicaciones están escritas en lenguaje de programación Java.
- **Marco de trabajo de aplicaciones**: los desarrolladores tienen acceso completo a los mismos APIs del framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades (sujeto a reglas de seguridad del framework). Este mismo mecanismo permite que los componentes sean remplazados por el usuario.
- **Bibliotecas**: Android incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema. Estas características se exponen a los desarrolladores a través del marco de trabajo de aplicaciones de Android; algunas son: System C library (implementación biblioteca C estándar), bibliotecas de medios, bibliotecas de gráficos, 3D y SQLite, entre otras.

- **Entorno de ejecución de Android**: Android incluye un conjunto de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Dalvik ha sido escrito de forma que un dispositivo puede correr múltiples máquinas virtuales de forma eficiente. Dalvik ejecuta archivos en el formato Dalvik Executable (.dex), el cual está optimizado para un consumo mínimo de memoria. La Máquina Virtual está basada en registros y corre clases compiladas por el compilador de Java que han sido transformadas al formato .dex por la herramienta incluida "dx".
- **Núcleo Linux**: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. El núcleo también actúa como una capa de abstracción entre el hardware y el resto de la pila de software.

2.3.2 Versiones de la plataforma

Se han realizado numerosas actualizaciones de la plataforma Android desde su lanzamiento original. Estas actualizaciones del sistema operativo normalmente solucionan algunos errores y añaden nuevas características. Generalmente, cada nueva versión de Android es desarrollada bajo un nombre en código basado en un postre o helado.
A continuación se explican muy brevemente las mejoras introducidas por las últimas actualizaciones de Android:

- **1.0**: liberado en septiembre de 2008.

- **1.1**: liberado el 9 de febrero de 2009.

- **1.5 (Cupcake)**: liberado en abril del 2009. Algunas de las mejoras introducidas fueron la posibilidad de grabar y reproducir videos. Capacidad de subir videos a YouTube e imágenes a Picasa.

- **1.6 (Donut)**: liberado en septiembre del 2009 y basado en el kernel de Linux 2.6.29. Se incluyó en esta actualización una experiencia mejorada en el Android Market y una interfaz mejorada de cámara, filmadora y galería.

- **2.0/2.1 (Eclair)**: liberados entre octubre de 2009 y enero de 2010 y basados en el kernel de Linux 2.6.69 introdujo una renovada interfaz de usuario así como HTML5 y soporte a **Exchange ActiveSync 2.5**.

- **2.2 (Froyo)**: liberado en mayo del 2010 y basado en el kernel de Linux 2.6.32 introdujo mejoras en la velocidad con la optimización JIT y el motor **Chrome** V8 Javascript. También añadía soporte para **Adobe** Flash y puntos de acceso WI-FI.

- **2.3 (Gingerbread)**: liberado en diciembre de 2010 y basado en el kernel de Linux 2.6.35.7 redefine la interfaz de usuario, mejoraba capacidades de teclado y funciones como **copy/paste** y añade soporte para **NFC (Near Field Communication)**.

- **3.0/3.1/3.2 (Honeycomb)**: liberada en febrero de 2011 se trató de una actualización orientada a *tabletas*. Soporta dispositivos con tamaños de pantalla mayor e introduce muchas características para la interfaz de usuario nuevas. Además, soporta procesadores multinúcleo y aceleración hardware para los gráficos. Mejoró el sistema multitarea y añadió soporte para gran cantidad de periféricos.

- **4.0 (Ice Cream Sandwich)**: versión que unifica el uso de cualquier dispositivo, tanto en teléfonos, tabletas, televisores, netbooks, etc. Opción de utilizar botones virtuales, en lugar de botones táctiles capacitivos. Aceleración hardware. Android Beam es la nueva característica que permite compartir contenido entre teléfonos. Vía NFC (Near Field Communication). Soporte nativo para el uso de Stylus (lápiz táctil).

2.3.2.1 Curiosidades con los nombres

Las versiones de Android reciben nombre de postres en inglés. En cada versión el postre elegido empieza por una letra distinta siguiendo un orden alfabético:

- C: Cupcake (v1.5), *magdalena glaseada*.
- D: Donut (v1.6), *rosquilla*.
- E: Éclair (v2.0/v2.1), pastel francés conocido en España como *pepito* o *canuto*.
- F: Froyo (v2.2), (abreviatura de «frozen yogurt») *yogur helado*.
- G: Gingerbread (v2.3), *pan de jengibre*.
- H: Honeycomb (v3.0/v3.1/v3.2), *panal*.
- I: Ice Cream Sandwich (v4.0), *sandwich de helado*.

NOTA: la aplicación desarrollada en este proyecto ha sido desarrollada y probada utilizando la versión 2.2 (API nivel 8) de Android.

2.3.2.2 Cuota de mercado

La compañía de investigación de mercado *Canalys* estimó que, en el segundo trimestre de 2009, Android tendría 2,8% del mercado de teléfonos inteligentes a nivel mundial.

En febrero de 2010, ComScore (Flosi, 2010) publicó que la plataforma Android tenía el 9% del mercado de teléfonos inteligentes en los Estados Unidos, según datos de los operadores. Esta cifra fue superior al estimado anterior de noviembre de 2009, el cual fue del 9%. Para finales del tercer trimestre de 2010, el mercado de Android en los Estados Unidos había crecido en un 21,4%.

En mayo de 2010, Android superó en ventas a iPhone, su principal competidor. Según un informe del grupo NPD (CNET, 2012), Android obtuvo un 28% de ventas en el mercado de los Estados Unidos, un 8% más que en el trimestre anterior. En el segundo trimestre de 2010, los dispositivos iOS incrementaron su participación en un 1%, indicando que Android está tomando mercado principalmente de RIM. Adicionalmente, los analistas apuntaron que las ventajas de que Android fuera un sistema multi-canal, multi-operador, le permitiría duplicar el rápido éxito que obtuvo el sistema Windows Mobile de Microsoft.

A principios de octubre de 2010, Google agregó 20 países a su lista de lugares geográficos donde los desarrolladores pueden enviar aplicaciones. Para mediados de octubre, la compra de aplicaciones estaba disponible en un total de 32 países. (Steven, 2010)

En diciembre de 2011 Andy Rubin (TICbeat, 2011) estimó que se activaban 700.000 dispositivos diariamente. Anteriormente en julio de 2011, el número de activaciones se estimaban en 550.000 dispositivos Android cada día. Se aprecia una tendencia de crecimiento en comparación con diciembre de 2010, 300.000 activaciones, y mayo de 2010, 100.000 activaciones.

2.3.2.3 Cuota de las versiones

Datos recogidos durante el período de 14 días que concluye el 5 de marzo del 2012. (Android Developers, Platform Versions, 2012)

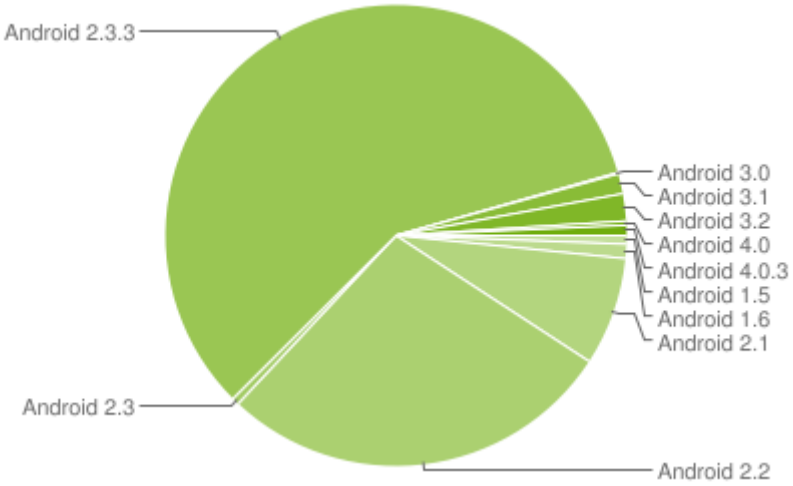


Ilustración 4. Cuota de mercado entre las diferentes versiones

2.3.3 La máquina virtual Dalvik

En Android, todas las aplicaciones se programan en el lenguaje Java y se ejecutan mediante una máquina virtual (VM) de nombre Dalvik (Code.google.com, Dalvik software, 2012), específicamente diseñada para Android. Esta máquina virtual ha sido optimizada y adaptada a las peculiaridades propias de los dispositivos móviles (menor capacidad de proceso, baja memoria, alimentación por batería, etc.) y trabaja con ficheros de extensión *.dex* (DalvikExecutables). Dalvik no trabaja directamente con el *bytecode* de Java, sino que lo transforma en un código más eficiente que el original, pensado para procesadores pequeños.

Gracias a la herramienta "dx", esta transformación es posible: los ficheros *.class* de Java se compilan en ficheros *.dex*, de forma que cada uno de ellos puede contener varias clase (Wikipedia, Dalvik (software), 2012) s. Después, este resultado se comprime en un único archivo de extensión *.apk* (Android Package), que es el que se distribuirá en el dispositivo móvil. Dalvik permite varias instancias simultáneas de la máquina virtual, y a diferencia de otras máquinas virtuales, está basada en registros y no en pila, lo que implica que las instrucciones son más reducidas y el número de accesos a memoria es menor.

La creación de una VM propia es un movimiento estratégico que permite a Google evitar conflictos con Sun por la licencia de la máquina virtual, así como asegurarse el poder innovar y modificar ésta sin tener que batallar dentro del JCP (Java Community Process).

2.3.3.1 Arquitectura

A diferencia de la mayoría de máquinas virtuales, (también Java VM), que están basadas en pila, la máquina virtual Dalvik **está basada en registro**.

Las ventajas de utilizar una arquitectura basada en pila frente a una basada en registro es objeto de debate. Generalmente, las máquinas virtuales basadas en pila usan instrucciones para cargar datos en la pila y manipularlos y, de este modo, se necesitan más instrucciones que en las máquinas virtuales basadas en registro para realizar el mismo código de alto nivel. Sin embargo, las instrucciones en una máquina virtual basada en registro deben contener tanto el registro origen como el destino por lo que generalmente son más largas. Esta diferencia es de una importancia básica para los intérpretes de las VM para quienes la ejecución de código máquina tiende a ser costoso junto con otros factores de igual relevancia como la compilación JIT (*Just in Time*). Análisis en profundidad (Slobojan, 2007) han concluido que las máquinas virtuales basadas en registro permiten una mayor optimización y por tanto una ejecución más rápida y son recomendadas para dispositivos limitados.

La herramienta **dx** es usada para convertir algunos (no todos) ficheros *.class* de Java al formato .dex. Se pueden incluir varias clases java en un único fichero *.dex*. Las cadenas u otras constantes que estén duplicadas solamente serán incluidas una vez en el fichero *.dex* para ahorrar espacio. El *java bytecode* también es convertido a un set de instrucciones usado por Dalvik.

Un fichero *.dex* sin comprimir suele ser más pequeño que un archivo java comprimido (*.jar*) proveniente del mismo fichero *.class*. Los ejecutables del Dalvik pueden ser modificados de nuevo cuando son instalados en un dispositivo móvil para hacer cierta optimización.

Desde la versión 2.2 de Android, Dalvik tiene compilación **JIT** (*Just in Time*).

Al haber sido optimizada para requerimientos de baja memoria, Dalvik tiene algunas características especiales que lo diferencian del resto de máquinas virtuales:

- La máquina virtual se adelgazó para utilizar menos espacio.
- El pool de constantes ha sido modificado para que use solamente índices de 32 bits, para simplificar el intérprete.
- Usa su propio *bytecode*, no el *bytecode* de JAVA.

Además, Dalvik fue diseñada para que un dispositivo pudiera ejecutar múltiples instancias de la máquina virtual de forma eficiente.

2.3.4 Estructura de una aplicación

Las Aplicaciones de Android están escritas en el lenguaje de programación Java. Las herramientas del Android SDK compilan el código junto con cualquier dato y archivos en un paquete de Android, un archivo con un sufijo .apk. Todo el código en un único archivo .apk es considerado como una única aplicación y es el archivo que los equipos que usan Android utilizan para instalar la aplicación. (Android Developers, Application Fundamentals, 2012)

Una vez instalada en el equipo, cada aplicación Android vive en su propio contenedor (*sandbox*):

- El sistema operativo Android es un sistema multi-usuario Linux en el que cada aplicación es un usuario diferente.
- Por defecto, el sistema asigna a cada aplicación una ID única de usuario (esta ID es usada únicamente por el sistema y es desconocida para la aplicación). El sistema le asigna permisos a todos los archivos en una aplicación para que solamente la ID de usuario asignada a esta aplicación pueda acceder a ellos.
- Cada proceso tiene su propia máquina virtual (MV) así que el código de una aplicación corre de manera aislada con respecto a otras aplicaciones.
- Por defecto, cada aplicación corre su propio proceso de Linux. Android comienza el proceso cuando cualquiera de los componentes de la aplicación necesita ser ejecutado, entonces apaga el proceso cuando ya no se necesita o cuando el sistema precisa recobrar memoria para otras aplicaciones.

De esta forma, el sistema Android implementa el principio del menor privilegio. De esta forma, cada aplicación, por defecto, tiene acceso únicamente a los componentes que requiere para hacer su trabajo y nada más. Esto crea un ambiente seguro en el cual una aplicación no puede acceder a partes del sistema para los que no se le ha dado permiso.

Sin embargo, hay maneras para que una aplicación comparta datos con otras o para que una aplicación pueda acceder a servicios de sistema.

- Es posible acordar que dos aplicaciones compartan la misma ID de usuario de Linux, en cuyo caso cada una podrá acceder a los archivos de la otra. Para conservar los recursos del sistema, las aplicaciones con la misma ID de Linux pueden correr bajo el mismo proceso y compartir la misma MV (las aplicaciones también deben compartir el mismo certificado).
- Una aplicación puede solicitar permiso para acceder a datos tales como contactos del usuario, mensajes SMS, el dispositivo de memoria, cámara, bluetooth y más. Todos los permisos de la aplicación deben ser concedidos por el usuario al momento de la instalación.

Esto cubre las bases acerca de como las aplicaciones de Android existen dentro del sistema. El resto del documento les introduce a:

- Los componentes principales del framework que definen tu aplicación.
- El archivo manifiesto en el cual se declaran los componentes y características del aparato que necesita la aplicación.
- Los recursos que están separados del código de la aplicación y le permiten de manera elegante optimizar su comportamiento para una variedad de configuraciones del aparato.

Los componentes de las aplicaciones son los bloques fundamentales de cualquier aplicación Android. Cada componente es un punto de acceso distinto para el sistema. No todos los componentes son puntos de entrada para el usuario y algunos dependen de otros, pero cada uno existe por sí mismo y juega un papel específico, cada uno es una parte que permite definir el comportamiento general de tu aplicación.

Existen cuatro tipos diferentes de componentes de aplicaciones. Cada tipo sirve a un propósito distinto, tiene un ciclo de vida que define como es creado y destruido. Aquí están los cuatro tipos de componentes de aplicaciones.

- ***Actividades*:** una Actividad o *Activity* representa una pantalla única con una interfaz de usuario. Por ejemplo, una aplicación de correo, puede tener una *Activity* que muestre una lista de nuevos mails, otra que permita escribir un nuevo correo y otra que permita leer mails. Aunque las *activities* trabajan juntas para formar la experiencia completa de trabajar con una aplicación de correo, cada una es independiente de las otras. De esta forma, una aplicación diferente puede iniciar cada una de estas *activities* (si la aplicación de mails lo permite). Por ejemplo, una aplicación de la cámara puede iniciar la *Activity* en la aplicación de correos que escribe un nuevo mail, para así poder compartir una fotografía.
- **Servicios:** un servicio es un componente que corre en el segundo plano para cumplir una operación de largo plazo o para cumplir el trabajo de procesos remotos. Un servicio no provee una interfaz de usuario. Por ejemplo, un servicio puede tocar música de fondo mientras el usuario se encuentra en una aplicación diferente, o puede obtener información de la red sin bloquear las interacciones del usuario con una *Activity*. Otro componente, como una *Activity*, puede iniciar un servicio y dejarlo corriendo o asociarse a él para interactuar.
- **Proveedores de Contenido**: un proveedor de contenido o *Content Provider* administra un conjunto de datos compartidos. Puedes guardar la información en el sistema de archivos, una base de datos SQLite, en la web o cualquier otro sistema de almacenamiento sistemático que tu aplicación pueda acceder. A través del proveedor de contenido, otras aplicaciones podrán buscar o incluso modificar la información (si el proveedor se los permite). Por ejemplo, El

sistema Android nos da un proveedor de contenido que administra la información de los contactos. Así, cualquier aplicación con los permisos necesarios, puede acceder a una parte del proveedor de contenido (tal como ContactsContract.Data) para leer y escribir información sobre una persona en particular.

Los proveedores de contenido también son útiles para leer y escribir información que es privada de tu aplicación y que no será compartida. Por ejemplo, el ejemplo de bloc de notas usa un proveedor de contenidos para guardar las notas.

- **Broadcast Receivers**: un broadcast receiver es un componente que responde a anuncios de sistema. Muchas emisiones se originan en el sistema, por ejemplo una emisión anuncia que la pantalla se ha apagado, la batería está baja o que se ha tomado una foto. Las aplicaciones también pueden iniciar emisiones, por ejemplo para hacer saber a otra aplicación que cierta información ha sido descargada y está disponible para su uso. Aunque los receptores de emisiones no muestran una interfaz de usuario, ellos pueden crear una notificación en la barra de estado para alertar al usuario que un evento de emisión está ocurriendo. Más comúnmente, sin embargo, un receptor de emisiones es sólo una puerta de ingreso a otros componentes, y se supone que realice una cantidad mínima de trabajo. Por ejemplo, puede iniciar un servicio para realizar un trabajo basado en el evento.

Un aspecto único del diseño del sistema Android es que cualquier aplicación puede iniciar un componente de otra aplicación. Por ejemplo, si se desea que el usuario tome una foto con la cámara del terminal, seguramente ya haya una aplicación que la puede realizar y no sería necesario implementar en todas las aplicaciones la utilización de la cámara, sería suficiente con realizar una llamada a la aplicación externa que se encargaría de hacerlo. No sería necesario hacer un link al código de la aplicación de la cámara, en vez de eso, simplemente se puede iniciar la *Activity* de la aplicación de la cámara que saca fotos. Cuando se haya terminado, la foto se puede devolver a la aplicación inicial para que se pueda trabajar en ella. Para el usuario, parecería que la cámara es parte de la aplicación.

Cuando el sistema inicia un componente, él comienza el proceso para esa aplicación (si no está corriendo ya) e instancia las clases necesarias para el componente. Por ejemplo, si la aplicación comienza la *Activity* de la cámara que saca una foto, esta *Activity* corre en el proceso que pertenece a la aplicación de la cámara, no en el proceso de la aplicación. Por tanto, a diferencia de aplicaciones en la mayoría de los otros sistemas, las aplicaciones de Android no tienen un único punto de ingreso (no hay función *main()*, por ejemplo).

El sistema ejecuta cada aplicación en un proceso separado con permisos restringidos para el acceso a otras aplicaciones, una aplicación no puede activar directamente un componente de otra aplicación. El sistema Android sin embargo, sí puede. Así que para activar un componente de otra aplicación, se debe enviar un mensaje al sistema que especifique el intento de iniciar un componente en particular. El sistema entonces activa el componente.

2.3.5 Componentes

En el siguiente apartado se definirán los componentes más importantes de la plataforma.

2.3.5.1 Activity

Una actividad o *Activity* es un componente de una aplicación que proporciona una pantalla con la cual los usuarios pueden interactuar con el fin de hacer algo, como marcar el teléfono, tome una foto, enviar un correo electrónico, o ver un mapa. A cada *Activity* se le da una ventana en la que se puede dibujar la interfaz de usuario. La ventana normalmente llena la pantalla, pero puede ser de un tamaño menor que el de la pantalla y flotar en la parte superior de otras ventanas (este es el caso de los Widgets que serán descritos posteriormente).

Una aplicación por lo general consiste de múltiples A*ctivities* que están más o menos ligadas entre sí. Por lo general, una *Activity* en una aplicación se especifica como la "principal" *Activity*, que se presenta al usuario al iniciar la aplicación por primera vez. Cada actividad puede iniciar otra *Activity* con el fin de realizar diferentes acciones. Cada vez que se comienza una nueva *Activity*, la *Activity* anterior se detiene, pero el sistema mantiene la *Activity* en una pila (la "pila de regreso"). Cuando se inicia una nueva *Activity*, esta se inserta en la parte superior de la pila y toma la atención del usuario. La pila sigue el mecanismo de colas llamado LIFO "el último en entrar, es el primero en salir", por lo que, cuando el usuario ha terminado con la actividad actual y presiona la tecla ATRÁS, se extrae de la pila (y es destruido) y se reanuda la actividad anterior.

Cuando una *Activity* se detiene debido a que una nueva *Activity* se inicia, este cambio es notificado a la *Activity* a través de los métodos de devolución de llamada (callback) de ciclo de vida de la *Activity*. Hay varios métodos de devolución de llamada que una *Activity* podría recibir, debido a un cambio en su estado, ya sea que el sistema la ha creado, detenido, reanudado, o destruido, y cada una de las devoluciones de llamadas te ofrece la oportunidad de realizar un trabajo específico que es apropiado para el cambio de estado. Por ejemplo, cuando se detuvo, su *Activity* debe liberar todos los objetos grandes, tales como conexiones de red o base de datos. Cuando la *Activity* se reanuda, se puede volver a adquirir los recursos necesarios y reanudar las acciones que

fueron interrumpidos. Estas transiciones de estado son parte del ciclo de vida de la *Activity.*

Ciclo de vida de una Activity

En una *Activity* existen fundamentalmente tres estados: *activa* o *en ejecución* (cuando se encuentra en el primer plano de la pantalla, siendo la *Activity* el objetivo de las acciones del usuario), en *pausa* (si se ha perdido el foco de acción, pero sigue siendo visible para el usuario) y *detenida* (si la *Activity* ha sido ocultada completamente por otra). Si una *Activity* está en pausa o detenida, el sistema puede terminar su ejecución, ya sea porque le pide finalizar o simplemente porque elimina su proceso, teniéndose en cuenta ciertos aspectos que serán comentados más adelante en esta sección.

La transición de un estado a otro se registra con las llamadas a los métodos *onCreate(), onStart(), onRestart(), onResume(), onPause(), onStop() y onDestroy().* Todos estos métodos se utilizan para realizar tareas específicas cuando cambia el estado de la *Activity*, como el caso de *onCreate()* el cual deben implementar todas las *Activities* y que se utiliza para aplicar la configuración inicial de una nueva instancia de la *Activity*, o el método *onPause()* que puede ser utilizado para guardar los cambios realizados sobre los datos.

En conjunto, estos métodos definen el ciclo de vida completo de una *Activity,* en el cual podemos diferenciar tres bucles de ejecución:

- **Ciclo de vida completo** de una *Activity* que sucede entre la primera llamada a *onCreate()* y la llamada final a *onDestroy().* La *Activity* lleva a cabo su configuración inicial en la llamada a *onCreate()*, y libera todos los recursos que quedan en o*nDestroy().*

- **Tiempo de vida visible** de una *Activity* que ocurre entre una llamada a *onStart()* hasta que se produce una llamada a *onStop().* Durante este tiempo, el usuario puede ver la *Activity* en pantalla, aunque puede que no sea en primer plano (interactuando con el usuario). Entre estos dos métodos, se pueden mantener los recursos que se necesitan para mostrar la *Activity* para el usuario.

- **Tiempo de vida en primer plano** de una *Activity* que ocurre entre una llamada a *onResume()* hasta que se produce una llamada a *onPause().* Durante este tiempo, la *Activity* se encuentra en primer plano y el usuario está interactuando con la misma.

El siguiente diagrama ilustra estos bucles y los caminos que una *Activity* puede tomar entre los distintos estados. Los óvalos de color son los estados más importantes

en los que se puede encontrar la *Activity*. Los rectángulos representan los métodos que se puede implementar para realizar operaciones cuando la *Activity* transita entre estados.

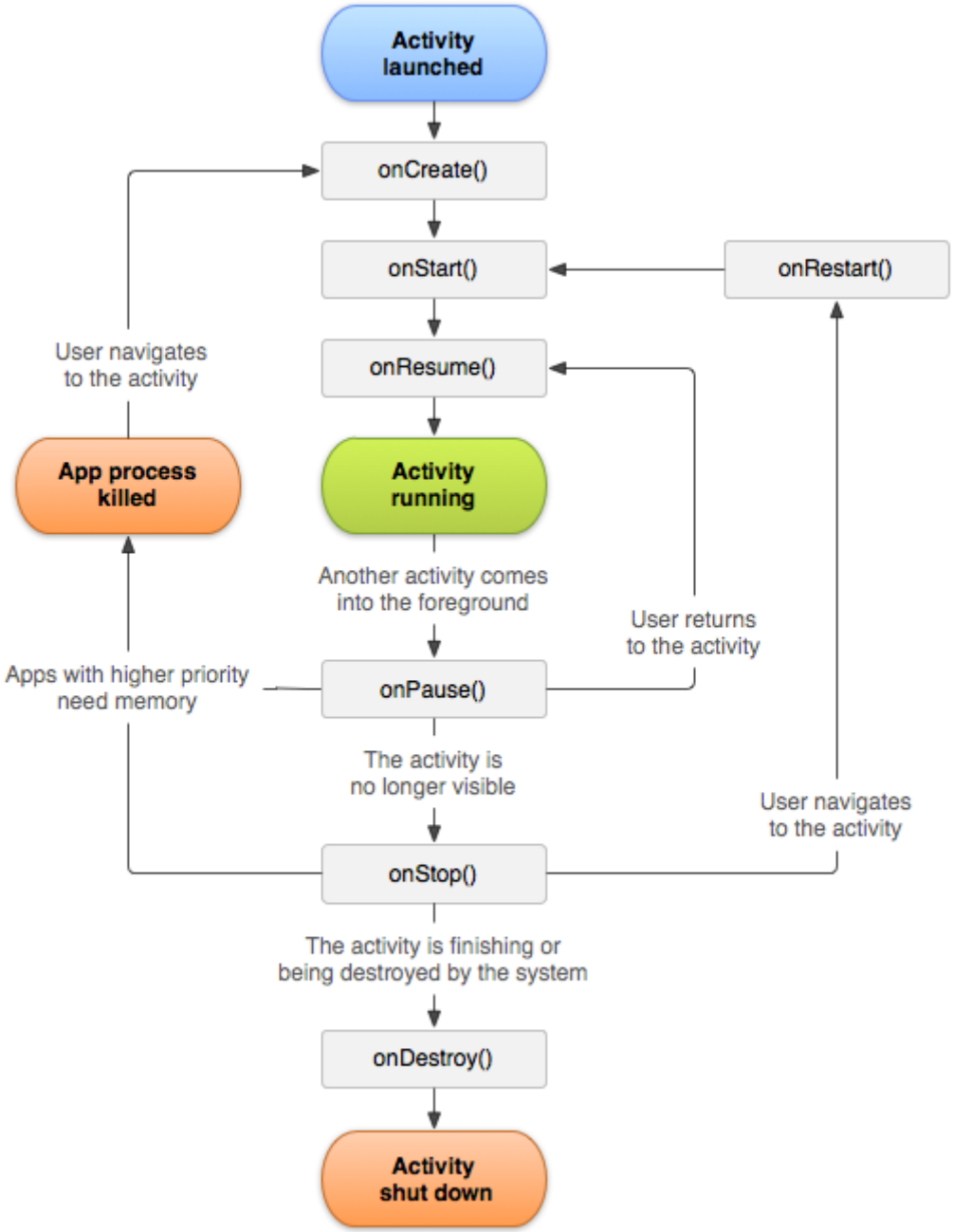


Ilustración 5. Ciclo de vida de una *Activity*

2.3.5.2 Servicio

Un servicio es un componente de aplicación que puede realizar operaciones de larga duración en segundo plano sin proporciona una interfaz de usuario. Otro componente de una aplicación puede iniciar un servicio y este seguirá funcionando en segundo plano, incluso si el usuario cambia a otra aplicación. Además, un componente

puede unirse a un servicio para interactuar con él e incluso realizar la comunicación entre procesos (IPC). Por ejemplo, un servicio puede manejar las transacciones de red, reproducir música, realizar operaciones con archivos de Entrada/Salida o interactuar con un proveedor de contenido, todo desde en segundo plano.

Un servicio esencialmente puede adoptar dos formas:

- **Iniciado (Started):** un servicio es "iniciado" cuando un componente de una aplicación (por ejemplo, una *Activity*) lo inicia llamando la función *StartService().* Una vez iniciado, un servicio puede ejecutarse en segundo plano de forma indefinida, incluso si el componente que empezó se destruye. Por lo general, un servicio iniciado realiza una sola operación y no devuelve un resultado a la persona que llama. Por ejemplo, puede descargar o cargar un archivo a través de la red. Cuando la operación se lleva a cabo, el servicio debe detenerse por si mismo.

- **Unido (Bound):** un servicio es "unido" cuando un componente de una aplicación se une a él llamando la función *bindService().* Un servicio unido ofrece una interfaz de cliente-servidor que permite a los componentes interactuar con el servicio, enviar solicitudes, obtener resultados, e incluso hacerlo a través de comunicación entre procesos (IPC). Un servicio unido sólo se ejecuta mientras que otro componente de la aplicación está vinculado a el múltiples componentes se pueden enlazar con un servicio a la vez, pero cuando todos ellos se desenlacen, el servicio se debe destruir a sí mismo.

Aunque los tipos de servicios se han presentado por separado, un servicio puede funcionar en ambos sentidos: se puede iniciar (para ejecutar de forma indefinida) y también permite la unión. Es simplemente una cuestión de implementar un par de métodos de devolución de llamada (*callbacks*): *onStartCommand()* para permitirle a un componente iniciarlo y *onBind()* para permitir la unión.

Cualquier componente de la aplicación puede utilizar un servicio, de la misma manera que cualquier componente puede utilizar una *Activity* haciendo uso de un objeto *Intent.* Es posible declarar los servicios como privados en el fichero de *manifest* bloqueando su uso a otras aplicaciones.

Ciclo de vida de un servicio

Los servicios se pueden usar de dos formas, dependiendo de como se lancen, su ciclo será uno u otro.

- Si se lanza con ***startService()*** se ejecutará hasta que termine. Los servicios se configuran en el método ***onCreate()*** y se liberan en el ***onDestroy()***. Podemos

terminar un servicio externamente con ***Context.stopService()*** o dentro del mismo servicio con ***Service.stopSelf() o Service.stopSelfResult().***

- Si se lanza con ***Context.bindService()*** podremos interactuar con él mediante una interfaz que el servicio debe exportar. Terminaremos el servicio con ***Context.unbindService().***

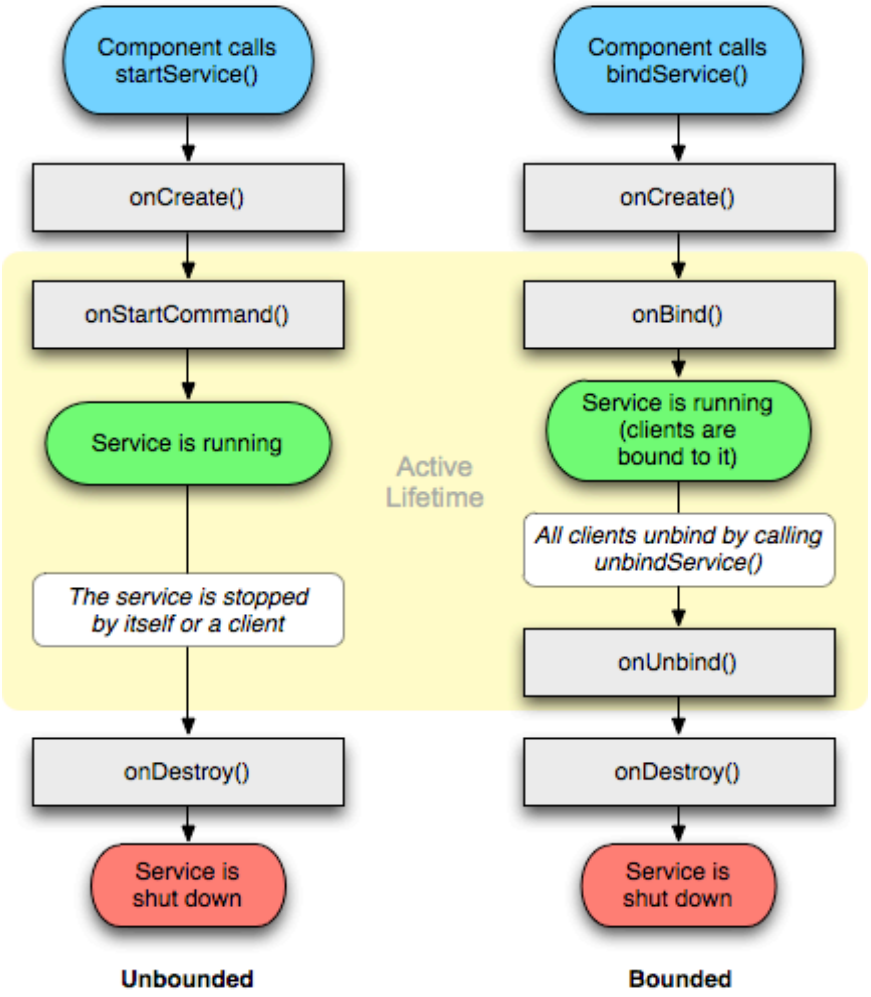


Ilustración 6. Ciclo de vida de un servicio en función del lanzamiento

2.3.5.3 Proveedores de contenido

Los proveedores de contenido (*Content Provider*) administran el acceso a un conjunto estructurado de datos. Encapsulan los datos, y proporcionan los mecanismos para definir la seguridad en los datos. Los proveedores de contenido son la interfaz estándar que conecta los datos de un proceso con el código que se esta ejecutando en otro proceso.

Cuando se desea acceder a los datos de un proveedor de contenido, se utiliza el objeto *ContentResolver* en el contexto de la aplicación para comunicarse con el proveedor como un cliente. El objeto *ContentResolver* se comunica con el objeto de proveedor, una instancia de una clase que implementa *ContentProvider*. El objeto de proveedor recibe las solicitudes de datos de clientes, lleva a cabo la acción solicitada, y devuelve los resultados.

No es necesario desarrollar un proveedor propio si no se tiene la intención de compartir datos con otras aplicaciones. Sin embargo, se necesita un proveedor propio para ofrecer sugerencias personalizadas de búsqueda. También se necesita un proveedor propio si se copian y pegan datos o archivos desde la aplicación a otras aplicaciones.

Android incluye su propio proveedor de contenido para administrar los datos, tales como audio, vídeo, imágenes e información de contacto personal. Se puede acceder a estos proveedores desde cualquier aplicación Android con algunas restricciones.

Ciclo de vida de un proveedor

Estos componentes tienen un sólo método asociado con el ciclo de vida que es *onReceive()*. Cuando llega un mensaje a un receptor de difusión se llama a este método pasándole dicho mensaje. En este momento, y sólo mientras ejecuta este método, el receptor se considera que está activo y por tanto un proceso que contenga a uno de estos componentes activos no podrá ser eliminado de la memoria del sistema.

2.3.5.4 Broadcast receivers

Un *Broadcast Receiver* es un componente que no hace nada excepto recibir y reaccionar ante anuncios. Muchos de estos anuncios los origina el propio sistema (por ejemplo, anuncios de que ha cambiado la zona horaria, de que la batería está baja o de que el usuario ha cambiado el idioma del sistema). Las aplicaciones también pueden crear anuncios como por ejemplo hacer saber a otras aplicaciones que se ha descargado algo o que el dispositivo está listo para usarse. (Android Developers, Broadcast Receiver, 2012)

Una aplicación puede tener cualquier número de componentes que respondan a los anuncios que se consideren importantes. Todos extenderán la clase *BroacastReceiver*. *BroadcastReceivers* no muestran una interfaz de usuario. Sin embargo, sí que pueden iniciar cualquier *Activity* como respuesta a un anuncio que hayan captado o pueden notificárselo al usuario a través de *NotificationManager*. Las notificaciones pueden captar la atención del usuario de diversas maneras: iluminando el dispositivo, haciéndolo vibrar, mediante sonidos, etc. Normalmente, mostrarán un icono en la barra de estado del dispositivo que el usuario podrá abrir para ver el mensaje de notificación.

Los *BroadcastReceivers* se pueden registrar dinámicamente dentro de la aplicación o declarando el registro con la etiqueta <receiver> en el fichero *AndroidManifest.xml*.

Si se registra un *BroadcastReceiver* en el método *onResume()* de una *Activity*, éste debería borrarse en el método *onPause()*, de este modo no se recibirán notificaciones cuando la *Activity* está pausada, evitando sobrecargas innecesarias. No se debe eliminar el registro de un BroadcastReceiver en el método *onSaveInstanceState()*, porque este método no será llamado si el usuario vuelve atrás en el historial.

Hay dos clases principales de emisiones (broadcast) que se pueden recibir:

- **Broadcast Normales**: enviados con *Context.sendBroadcast()*, son completamente asíncronos. Todos los receptores del *broadcast* se ejecutan en un orden sin definir, con frecuencia al mismo tiempo. Esto es más eficiente, pero significa que los receptores no pueden usar el resultado.
- **Broadcast Ordenados**: enviados con *Context.sendOrderedBroadcast()*, se entregan a un receptor cada vez. A medida que cada receptor se ejecuta, éste puede propagar un resultado al siguiente receptor o puede abortar el *broadcast* por completo de modo que no será propagado a ningún otro receptor. El orden en que se ejecuta cada receptor puede ser configurado con el atributo *android:priority*.

Incluso en el caso de los *broadcast* normales, en ciertas ocasiones el sistema puede decidir entregar el *broadcast* a los receptores de uno en uno. Particularmente, si los receptores requieren la creación de un proceso, solamente se ejecutará uno cada vez para evitar sobrecarga en el sistema con los nuevos procesos. En esta situación, sin embargo, se mantienen las restricciones de los *broadcast* normales: estos receptores no podrán devolver resultados ni abortar el *broadcast*.

Nótese que, aunque se está usando la clase *Intent* para el envío y recepción de *broadcasts*, este mecanismo es completamente distinto al usado para iniciar *Activities* con el método *Context.startActivity()*. No hay forma de que un BroadcastReceiver pueda ver o capturar *Intents* usados con *startActivity*(). Igualmente, cuando se hace una difusión de un *Intent*, nunca se encontrará un iniciará una *Activity*. Estas dos operaciones son semánticamente muy distintas: iniciar una *Activity* con un *Intent* es una operación en primer plano que modifica aquello con lo que el usuario está interactuando, mientras que difundir un *Intent* es una operación en segundo plano de la que el usuario normalmente no tiene constancia.

Ciclo de vida de un receptor

Un objeto BroadcastReceiver solamente es válido mientras dure la llamada al método *onReceive(Context, Intent)*. Una vez que el código vuelve de esta función, el sistema considera que el objeto ya está terminado e inactivo. Esto tiene importantes repercusiones sobre lo que se puede hacer en la implementación del método *onReceive(Context, Intent)*: cualquier cosa que requiera operaciones asíncronas no está disponible, dado que se necesitaría volver de la función para manejar la operación asíncrona, pero en ese punto el BroadcastReceiver ya no está activo por lo que el sistema sería libre de matar su proceso antes de que la operación asíncrona terminase.

En particular, no se puede mostrar un diálogo o enlazarse a un servicio desde dentro de un BroadcastReceiver. Para los primeros, se debería utilizar el API de NotificationManager en su lugar. Para los últimos, se puede utilizar *Context.startService()* para mandar un comando a un servicio.

Ciclo de vida de un proceso

Un proceso que está ejecutando actualmente un BroadcastReceiver (esto es, ejecutando código en su método *onReceive(Context, Intent)*), es considerado un proceso en primer plano, y seguirá siendo ejecutado por el sistema excepto en casos extremos de escasez de memoria.

Una vez que se vuelve de *onReceive()*, el BroadcastReceiver no está activo y el proceso donde se ejecuta es igual de importante como lo sea cualquier otro componente de la aplicación que se esté ejecutando en él. Esto es especialmente importante porque si en el proceso solamente se estaba ejecutando el BroadcastReceiver, entonces a la vuelta del *onReceive()* el sistema considerará que el proceso está vacío y, por tanto, lo matará para liberar recursos que puedan ser utilizados por procesos más importantes.

Esto significa que para operaciones de larga duración se usará frecuentemente un servicio junto con un BroadcastReceiver para mantener el proceso contenedor activo durante todo el periodo que dure la operación.

2.3.5.5 Intents

Un *Intent* es una descripción abstracta de una operación que se va a realizar. Puede utilizarse para lanzar *Activity* es, enviar *broadcasts* a *BroadcastReceivers*, iniciar servicios o comunicarse con servicios en ejecución. (Android Developers, Intent, 2012)

Un *Intent* proporciona facilidades para enlazar código en diferentes aplicaciones en tiempo de ejecución. Su uso más significativo es para lanzar *Activities*, donde se puede considerar como el pegamento entre *activities*. Básicamente, se trata de una estructura de datos pasiva que contiene una descripción abstracta de la acción que se quiere realizar. Las principales piezas de información de un *Intent* son:

- **Action**: la acción general que se va a realizar, tales como ACTION VIEW, ACTION EDIT, ACTION MAIN, etc.
- **Data**: los datos sobre los que operar, tales como contactos, personas, expresados como un URI (*Uniform Resource Identifier*).

Además de estos atributos primarios, existen otros secundarios que se pueden incluir con un *Intent*:

- **Category**: da información adicional acerca de la acción a ejecutar.
- **Type**: especifica el tipo de datos del *Intent*. Normalmente el tipo se infiere de los datos mismos. Configurando este atributo, se desactiva la evaluación y se fuerza un tipo explícito.
- **Component**: especifica el nombre explícito de la clase de componente que se usa para el *Intent*. Normalmente esta información se determina mirando el resto de información del *Intent*. Si este atributo está configurado entonces toda la evaluación previa se anula. Especificando este atributo convierte al resto en opcionales.
- **Extras**: esto es un objeto *Bundle* (objeto en el que se pueden almacenar todo tipo de valores con estructura *clave-valor*) con cualquier tipo de información adicional. Se puede utilizar para proporcionar información extendida al componente.

Hay diferentes métodos para activar cada tipo de componente: (Android Developers, Activating Components, 2012)

Una *Activity* puede ser lanzada llamando a los métodos *Context.startActivity()* o *Activity.startActivityForResult()*, pasándole a cada uno de ellos un objeto *Intent*. La *Activity* lanzada puede obtener la información acerca del *Intent* inicial llamando al método *getIntent()*.

Con frecuencia, una *Activity* lanza la siguiente. Si la primera espera un resultado de la segunda, llamará al método *startActivityForResult()* en vez de *startActivity()*. Por ejemplo, si se lanza una *Activity* para que el usuario seleccione una foto, se podría esperar que devuelva la foto que ha sido elegida. El resultado es devuelto en un objeto *Intent* pasado al método *onActivityResult()* de la *Activity* que inició la llamada.

Un servicio es iniciado mediante la llamada al método *Context.startService()*, pasándole un objeto *Intent*. Android llamar á al método *onStart()* del servicio y le pasará dicho objeto *Intent*.

De forma similar, se puede llamar al método *Context.bindService()*, pasándole el objeto *Intent*, para establecer una conexión entre el componente de llamada y un servicio destino. El servicio recibe el objeto *Intent* en una llamada al método *onBind()*.

Por ejemplo, una *Activity* podría establecer una conexión con el servicio de reproducción de música para ofrecer al usuario una interfaz con la que manejar tal reproductor. La *Activity* llamaría al método *bindService* para establecer esa conexión y luego llamar a los métodos definidos por el servicio que puedan afectar a la reproducción.

Un aplicación puede iniciar una emisión pasando un objeto *Intent* a métodos como *Context.sendBroadcast()*, *Context.sendOrderedBroadcast()*, y *Context.sendStickyBroadcast()* , en cualquiera de sus variantes. Android entrega el *Intent* a todos los recibidores de emisiones interesados llamando a sus métodos *onReceive*.

Hay dos formas principales de *Intents* que un usuario utilizará (Android Developers, Intent Resolution, 2012):

- Los **Intents explícitos** han especificado un componente (usando el método *setComponent(componentName)* o *setClass(Context,Class)*), lo que proporciona la clase exacta a ser ejecutada. Con frecuencia esto no incluirá ninguna otra información, siendo simplemente una forma de lanzar Activities por parte de la aplicación.
- Los **Intents implícitos** no han especificado ningún componente. En vez de eso, deben incluir suficiente información para que el sistema determine cual de los componentes disponibles es el mejor para ejecutar la acción descrita en el *Intent*.

Cuando se usan *Intents* implícitos, se tiene que saber qué hacer con ellos.
Esto es manejado por el proceso *Intent Resolution* (Resolución de *Intents*), el cuál mapea un *Intent* a una *Activity*, BroadcastReceiver o servicio.

El mecanismo de Resolución de *Intents* básicamente lo que hace es comparar un *Intent* con todas las líneas <intent-filter> en los paquetes de la aplicación instalados (además, en caso de broadcasts, busca en todos los objetos registrados con el método *registerReceiver(BroadcastReceiver, IntentFilter)*).

Un Content Provider está activo únicamente cuando está respondiendo a una petición proveniente de un objeto ContentResolver. Y un BroadcastReceiver está activo sólo cuando está respondiendo a un mensaje broadcast. Así que no hay una necesidad de cerrar esos componentes explícitamente. Por otra parte, las *activities* proporcionan la interfaz de usuario. Están en una larga conversación con el usuario y pueden permanecer activas, incluso si no están siendo utilizadas, mientras la conversación continúa. De forma similar, los servicios pueden permanecer ejecutándose un largo periodo de tiempo. Así que Android tiene métodos para cerrar *activities* y servicios de forma ordenada:

- Una *Activity* puede ser cerrada llamando a su método *finish()*. Una *Activity* puede cerrar otra *Activity* llamando al método *finishActivity()*.
- Un servicio puede ser parado llamando a su método *stopSelf()* o llamando al método *Context.stopService()*.

Los componentes también podrían ser cerrados por el sistema cuando éstos no vayan a ser usados más o cuando Android reclame memoria para otros componentes activos.

2.3.5.6 Fichero Android Manifest

Toda aplicación Android ha de tener un fichero *AndroidManifest.xml* (exactamente con ese nombre) en su directorio raíz. El manifiesto presenta información esencial acerca de la aplicación al sistema Android, información que el sistema debe tener antes de que pueda ejecutar el código de cualquier aplicación. Entre otras cosas, el fichero hace lo siguiente: (Android Developers, The AndroidManifest.xml File, 2012)

- Le da nombre al paquete JAVA de la aplicación. Este nombre sirve como identificador único para la aplicación.
- Describe los componentes de la aplicación; *activities*, servicios, *broadcast receivers* y cualquier *content provider* que contenga la aplicación. Le da nombre a las clases que implementan cada uno de los componentes y publica sus capacidades (por ejemplo, qué mensajes pueden manejar). Estas declaraciones permiten conocer al sistema Android cuáles son los componentes y bajo qué condiciones pueden ser lanzados.
- Determina qué procesos albergarán los componentes de la aplicación.
- Declara qué permisos debe tener la aplicación de modo que pueda acceder a partes protegidas del API e interactuar con otras aplicaciones.
- También declara qué permisos deben tener otras aplicaciones para que éstas puedan interactuar con los componentes de la aplicación.
- Lista las clases de instrumentación que proporcionan perfiles y otra información mientras la aplicación se está ejecutando. Estas declaraciones sólo tienen sentido

cuando la aplicación está siendo desarrollada y probada. Finalmente son borradas antes de que la aplicación sea publicada.

- Declara el mínimo nivel de API que requiere la aplicación de Android.
- Lista las librerías contra las que la aplicación debe ser enlazada.

2.3.5.7 Interfaz de usuario

En Android la interfaz de usuario se construye básicamente con dos componentes: *View* y *ViewGroup*. (Android Developers, User Interface, 2012)

Cada elemento *View* contenido en la interfaz de usuario almacena información acerca de la disposición y el contenido de una región específica de la pantalla, gestionando distintos aspectos como el tamaño, los cambios en el foco de atención o las interacciones con esa zona específica de la pantalla por parte del usuario. Los objetos *View* sirven de base a otros elementos llamados *widgets*, los cuales ofrecen funcionalidades ya implementadas y que pueden ser utilizadas por la interfaz de usuario de una *Activity*, como por ejemplo un campo de texto o un botón. Por su parte los objetos *ViewGroup* sirven para definir la disposición o *layout* de los elementos que componen la interfaz.

Para definir la interfaz de usuario se debe establecer una jerarquía de objetos *View* y *ViewGroup*, como por ejemplo la mostrada en la figura de la página siguiente, utilizando para ello un conjunto de *widgets* y *layouts*, que pueden estar predefinidos o creados desde cero por el desarrollador.

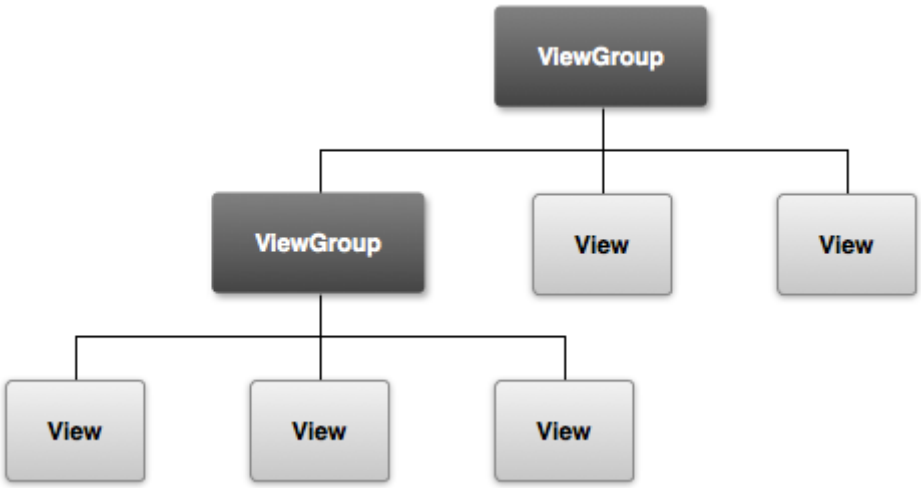


Ilustración 7. Jerarquía que define una interfaz de usuario

La forma más común de establecer esta jerarquía es con un archivo XML que contenga la disposición de los distintos elementos. Cada elemento especificado en este archivo es un objeto *View* o *ViewGroup* (o desciende de alguno de ellos), siendo los nodos hoja del árbol que representa esta jerarquía los objetos *View* y las ramas del mencionado árbol los objetos *ViewGroup*. Cada elemento tiene un nombre distintivo

que representa una clase de Java que lo implementa. Por ejemplo un elemento <*TextView*> crea un objeto *TextView* en la interfaz, y un elemento <*LinearLayout*> crea un objeto ViewGroup equivalente. Por ejemplo, el siguiente fragmento de código crearía un ViewGroup en el que sus elementos, un cuadro de texto y un botón, se mostrarán uno a continuación del otro:

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
              android:layout_width="fill_parent"
              android:layout_height="fill_parent"
              android:orientation="vertical" >
      <TextView android:id="@+id/text"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Hello, I am a TextViev" />
      <Button    android:id="@+id/button"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Hello, I am a Button" />
</LinearLayout>
```

Ejemplo 1. Codificación de un *ViewGroup*

En el ejemplo anterior se utilizan dos *widgets* predefinidos de Android, *TextView* y *Button* que como se comentó anteriormente son objetos de tipo *View* que sirven de interfaz para la interacción con el usuario. Las acciones llevadas a cabo son registradas por cada objeto *View* mediante una de las dos formas siguientes:

- Estableciendo “oyentes” dentro de cada objeto *View* que se dedique a manejar las acciones del usuario, como por ejemplo cuando se pulse un botón se definirá un método *OnClicklistener()* que realizará una acción determinada.
- Redefiniendo los eventos desencadenados por la acción del usuario (cuando se implementa un nuevo objeto *View*).

Otro aspecto importante en una interfaz de usuario son los menús, ya que ofrecen una interfaz sencilla para realizar alguna acción dentro de la aplicación. La forma más común de mostrar un menú en Android es presionando el botón de menú con el que cuentan todos los dispositivos diseñados para la plataforma, aunque también es habitual mostrar menús contextuales a través de elementos de la interfaz.

Los menús al igual que los elementos que componen la interfaz de usuario se estructuran de forma jerárquica, pero normalmente son creados de forma distinta a dicha jerarquía, pues se definen en los métodos *onCreateOptionsMenu()* y *onCreateContextMenu( )* dentro de las *Activities*. Estos componentes de la interfaz de usuario manejan también sus propios eventos, para poder capturar las diferentes acciones.

Por último, mencionar que Android permite una completa personalización de la interfaz de usuario pudiéndose aplicar estilos y temas visuales a los elementos de una aplicación, ya sean creados por el usuario o de los predefinidos por el sistema.

Capítulo III. Tecnologías y herramientas

3.1 Tecnologías utilizadas

En el presente capítulo se presenta el análisis de las herramientas y tecnologías empleadas en la elaboración de este proyecto.

3.1.1 SQlite

SQLite es un sistema de bases de datos relacional (Sqlite.org, 2012) , contenida en una biblioteca en C. SQLite es un proyecto de dominio público creado por D. Richard Hipp. A diferencia de los sistemas de gestión de bases de datos cliente-servidor, el motor de SQLite no es un proceso independiente con el que el programa principal se comunica. En lugar de eso, la biblioteca SQLite se enlaza con el programa pasando a ser parte integral del mismo.

Ilustración 8. Logo de SQLite

El programa utiliza la funcionalidad de SQLite a través de llamadas simples a subrutinas y funciones. Esto reduce la latencia en el acceso a la base de datos, debido a que las llamadas a funciones son más eficientes que la comunicación entre procesos. El conjunto de la base de datos (definiciones, tablas, índices, y los propios datos), son guardados como un sólo fichero estándar en la máquina host. Este diseño simple se logra bloqueando todo el fichero de base de datos al principio de cada transacción.

En su versión 3, SQLite permite bases de datos de hasta 2 Terabytes de tamaño, y también permite la inclusión de campos tipo BLOB.

El gestor de base de datos por defecto de Android es Lite. SQLite es una base de datos transaccional ligera que ocupa una cantidad muy pequeña de espacio en disco y memoria, de manera que es la elección perfecta para crear bases de datos en sistemas operativos para móviles como Android o iOS.

Aspectos a tener en cuenta cuando se maneja SQLite:

- SQLite no dispone de control de integridad, por lo que se puede almacenar un valor de un cierto tipo en un campo de otro tipo distinto (por ejemplo se puedo poner un *String* o un *Integer* en un mismo campo).
- SQLite no gestiona directamente la integridad referencial, no soporta restricciones *FOREING KEY* (clave externa). No obstante se puede controlar mediante *triggers*.
- El soporte completo de Unicode es opcional y no esta instalado por defecto.

3.1.2 XML

XML, siglas en inglés de *eXtensible Markup Language* (lenguaje de marcas extensible), es un metalenguaje extensible de etiquetas desarrollado por el W3C (*World Wide Web Consortium*) (W3C, XML, 2012). Es una simplificación y adaptación del SGML y permite definir la gramática de lenguajes específicos (de la misma manera que HTML es a su vez un lenguaje definido por SGML).

Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades, de ahí que se le denomine metalenguaje. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML no ha nacido sólo para su aplicación en Internet, sino que se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

3.1.3 Java

Sun Microsystems desarrolló, en 1991, el lenguaje de programación orientado a objetos que se conoce como Java (Java.com, 2012). El objetivo era utilizarlo en un *set-top box*, un tipo de dispositivo que encarga de la recepción y la decodificación de la señal televisiva. El primer nombre del lenguaje fue Oak, luego se conoció como Green y finamente adoptó la denominación de Java.

Ilustración 9. Logotipo de Java

La intención de Sun era crear un lenguaje con una estructura y una sintaxis similar a C y C++, aunque con un modelo de objetos más simple y eliminando las herramientas de bajo nivel.

Los pilares en los que se sustenta Java son cinco: la programación orientada a objetos, la posibilidad de ejecutar un mismo programa en diversos sistemas operativos, la inclusión por defecto de soporte para trabajo en red, la opción de ejecutar del código en sistemas remotos de manera segura y la facilidad de uso.

Lo habitual es que las aplicaciones Java se encuentren compiladas en un *bytecode* (un fichero binario que tiene un programa ejecutable), aunque también pueden estar compiladas en código máquina nativo.

Sun controla las especificaciones y el desarrollo del lenguaje, los compiladores, las máquinas virtuales y las bibliotecas de clases a través del Java Community Process. En los últimos años, la empresa (que fue adquirida por Oracle) ha liberado gran parte de las tecnologías Java bajo la licencia GNU GPL.

El lenguaje de programación Java, es un lenguaje orientado a objetos que ofrece una serie de ventajas a los desarrolladores:

- Java es simple: fue diseñado para ser fácil de usar y en consecuencia es más fácil de escribir, compilar, depurar y que otros lenguajes de programación. La razón por la que Java es mucho más sencillo que C++ es porque Java utiliza un sistema automático para la gestión del uso de la memoria (recolector de basura) mientras que en C++ esta gestión la ha de hacer el programador.
- Java está orientado a objetos: programar en Java está centrado en crear objetos, manipular objetos y hacer que éstos se intercambien mensajes. Esto permite crear programas modulares fácilmente reusables.
- Es independiente de la plataforma: una de las ventajas más significativas de Java es la facilidad que ofrece para ser portado de una plataforma a otra. Esta habilidad para ejecutar el mismo programa en muchos sistemas diferentes es crucial para las aplicaciones de la WWW(*World Wide Web*), y en este sentido Java lo hace muy bien, siendo independiente de la plataforma tanto a nivel de

código fuente como al de los binarios.

- Java es un lenguaje interpretado: se necesita un intérprete para ejecutar programas hechos con Java. Los programas son compilados generando un código denominado *bytecode*, que será utilizado por la máquina virtual de java para ejecutarlo. Esto implica además que un programa puede ser compilado sólo una vez y el *bytecode* generado puede ser ejecutado en diferentes plataformas.
- Java es distribuido: la computación distribuida implica tener muchos ordenadores en una red trabajando juntos. Java está diseñado para hacer esta computación distribuida sencilla con las capacidades de red que están inherentemente incluidas en ella.
- Java es seguro: Java es uno de los primeros lenguajes de programación en considerar la seguridad como parte de su diseño. El lenguaje, compilador, intérprete y entorno de ejecución fueron cada uno desarrollados teniendo en cuenta la seguridad.
- Es robusto: Java pone mucho énfasis en comprobar posibles errores con la mayor antelación posible. Es por ello que los compiladores son capaces de detectar muchos de los problemas que en otros lenguajes de programación se verían en tiempo de ejecución.
- Es multitarea: en Java, la programación multi hilo ha sido suavemente integrada, mientras que en otros lenguajes se han de ejecutar procedimientos dependientes del sistema operativo para realizar estas tareas multi hilo.

Las aplicaciones de Android están escritas en el lenguaje de programación Java aunque, como se ha visto en anteriores apartados, éstas son ejecutadas en su propia máquina virtual (Dalvik).

Ilustración 10. Logotipo de Android

3.1.4 HTTP

HTTP, siglas en inglés *Hypertext Transfer Protocol* (en español *protocolo de transferencia de hipertexto*) (W3C, HTTP - Hypertext Transfer Protocol, 2012)es el protocolo usado en cada transacción de la World Wide Web. HTTP fue desarrollado por el *World Wide Web Consortium* y la *Internet Engineering Task Force*, colaboración que culminó en 1999 con la publicación de una serie de *RFC*, el más importante de ellos es el *RFC* 2616 que especifica la versión 1.1. HTTP define la sintaxis y la semántica que utilizan los elementos de software de la arquitectura web (clientes, servidores, proxis) para comunicarse. Es un protocolo orientado a transacciones y sigue el esquema petición-respuesta entre un cliente y un servidor. Al cliente que efectúa la petición (un navegador web o un spider) se lo conoce como "*user agent*" (agente del usuario). A la información transmitida se la llama recurso y se la identifica mediante un localizador uniforme de recursos (URL). Los recursos pueden ser archivos, el resultado de la ejecución de un programa, una consulta a una base de datos, la traducción automática de un documento, etc.

HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores. El desarrollo de aplicaciones web necesita frecuentemente mantener estado. Para esto se usan las cookies, que es información que un servidor puede almacenar en el sistema cliente. Esto le permite a las aplicaciones web instituir la noción de “sesión”, y también permite rastrear usuarios ya que las cookies pueden guardarse en el cliente por tiempo indeterminado.

La típica transacción de protocolo HTTP se compone de un encabezado seguido por una línea en blanco y luego un dato. Este encabezado define la acción requerida por el servidor.

Desde su creación, el HTTP evolucionó en diversas versiones. Entre ellas, la 0.9, la 1.0, la 1.1 y la 1.2. Este protocolo trabaja con códigos de respuesta de tres dígitos, que comunican si la conexión fue rechazada, si se realizó con éxito, si ha sido redirigida hacia otro URL, si existe un error por parte del cliente, o bien, por parte del servidor.

3.1.5 ASP. NET

ASP.NET es un *framework* (marco de trabajo) para aplicaciones web desarrollado y comercializado por Microsoft. Suele utilizarse para construir sitios web dinámicos, aplicaciones web y servicios web XML (Microsoft, 2012). Apareció en enero de 2002 con la versión 1.0 del .NET Framework, y es la tecnología sucesora de la tecnología *Active Server Pages* (ASP). ASP.NET está construido sobre el *Common Language*

Runtime, permitiendo a los programadores escribir código ASP.NET usando cualquier lenguaje admitido por el .NET Framework.

Ilustración 11. Logotipo de ASP.NET

El principio de la tecnología ASP es el VBScript, pero existe otra diversidad de lenguajes de programación que pueden ser utilizados como lo es Perl, JScript, etc.

El ASP es una tecnología dinámica funcionando del lado del servidor, lo que significa que cuando el usuario solicita un documento ASP, las instrucciones de programación dentro del script son ejecutadas para enviar al navegador únicamente el código HTML resultante. La ventaja principal de las tecnologías dependientes del servidor radica en la seguridad que tiene el programador sobre su código, ya que éste se encuentra únicamente en los archivos del servidor que al ser solicitado a través del web, es ejecutado, por lo que los usuarios no tienen acceso más que a la página resultante en su navegador.

3.2 Herramientas

A continuación se detallan las herramientas que se han utilizado para el desarrollo de este proyecto así como la configuración necesaria llevada a cabo en dichas herramientas.

3.2.1 Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar lo que el proyecto llama “Aplicaciones de Cliente Enriquecido”, opuesto a las aplicaciones “Cliente-liviano" basadas en navegadores (Eclipse, Eclipse Project, 2012). Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrados (del inglés IDE), como el IDE de Java llamado JDT (*Java Development Toolkit*) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse).

Ilustración 12. Logotipo de eclipse

Eclipse es una herramienta muy adecuada para el desarrollo de aplicaciones para Android. La propia Google ha impulsado y promovido el uso de Eclipse, desarrollando un *plugin* exclusivo para ello llamado ADT (Android Development Tool). ADT extiende las capacidades de Eclipse de modo que permite crear proyectos Android de manera rápida y sencilla, ejecutar y depurar las aplicaciones utilizando las herramientas del SDK de Android e incluso generar los archivos *.apk* firmados para poder distribuirlos. Por estas razones, se ha decidido utilizar Eclipse para el desarrollo del proyecto.

3.2.1.1 Instalación de eclipse

La instalación de eclipse se realiza de una forma sencilla, se puede descargar de la página de eclipse (Eclipse, Eclipse Project, 2012) en forma de archivo ZIP y solo se tendría que descomprimir en la carpeta donde se quiera tenerlo instalado. Para ejecutarlo solo hay que arrancar el fichero *Eclipse.exe* . Una vez arrancado lo único que se ha de configurar es la ruta por defecto donde se quieren guardar los proyectos también denominado *workspace*.

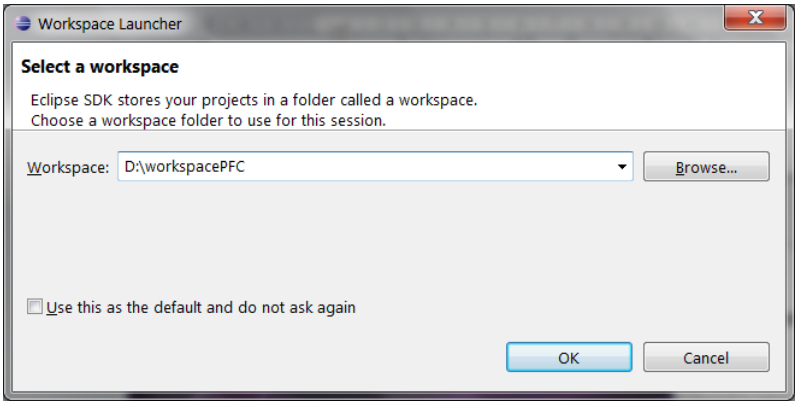


Ilustración 13. Selección del workspace en eclipse

3.2.2 SDK Android

Antes de empezar con la instalación de la plataforma, es necesario comprobar que se cumplen los requisitos mínimos del sistema. Una vez confirmado este punto, es necesario descargar e instalar el SDK de Android. (Android Developers, System Requirements, 2012)

3.2.2.1 Instalación del SDK de Android

En la página de descargas de Android (Android Developers, Download the Android SDK, 2012) se puede obtener la última versión del SDK (versión 4.0.3 a fecha de Mayo de 2012), en versión ejecutable o comprimida en un fichero .zip. Una vez obtenido el SDK, bastará con descomprimirlo en la ruta que el usuario prefiera. Esta ruta será utilizada posteriormente para hacer referencia al SDK, por lo que será necesario recordarla.

3.2.2.2 Instalación de ADT

ADT es un *plugin* para el entorno de desarrollo Eclipse. Extiende las capacidades de Eclipse de modo que, utilizando las herramientas proporcionadas por el SDK de Android, permite configurar rápidamente proyectos, crear interfaces de usuario y depurar las aplicaciones Android.

Antes de poder instalar ADT, es necesario tener instalada una versión compatible de Eclipse. Si no es el caso, se puede descargar desde la página oficial de descargas de Eclipse, (Eclipse, Eclipse Downloads, 2012).

Descarga del plugin ADT

A continuación se detallan los pasos necesarios para instalar el plugin ADT en el entorno Eclipse, se puede instalar directamente desde la interfaz de eclipse.

Desde la barra de herramientas de la pantalla principal de eclipse ***Help →Install New Software… → Add***. *E*n el diálogo de ***Add site*** se configura:
Name**:** *Android plugin*
Location**:** *https://dl-ssl.google.com/android/eclipse/*

En este momento eclipse buscará las posibles actualizaciones. La más importante sería marcar la ***Android Developer Tools***, se deben marcar e instalar. Una vez instalado el puglin de eclipse pedirá reiniciar y aparecerán las herramientas de Android desde la interfaz de eclipse.

Configuración de ADT

Una vez reiniciado Eclipse, desde ***Windows → Preferences***, y en el diálogo que abre, en el apartado ***Android***, se rellena el campo ***SDK Location***. Ya sea introduciendo la ruta a mano o mediante el botón ***Browser***, se introduce la ruta donde se ha descomprimido el fichero del apartado 3.2.2.1.

Si todo ha ido correctamente, la instalación del plugin ya estaría terminada.

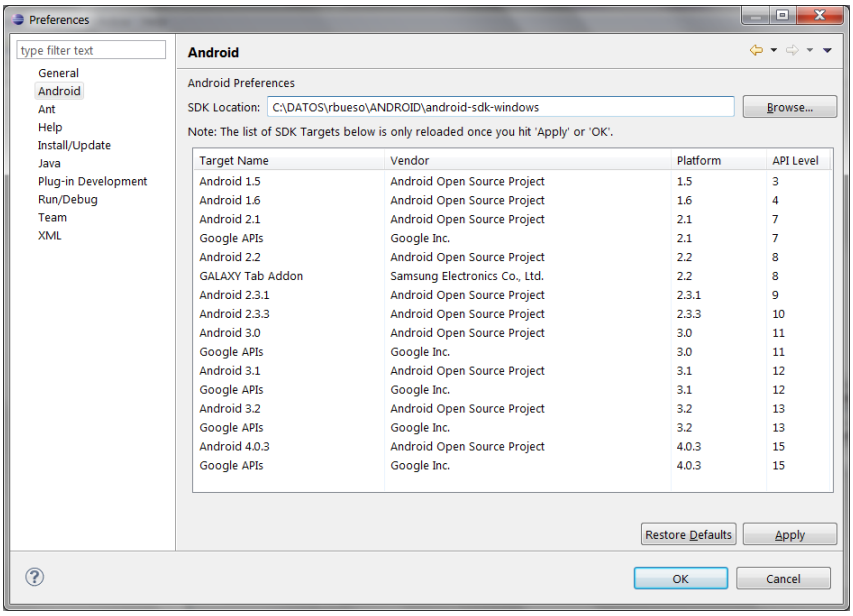


Ilustración 14. Configuración del plugin ADT

3.2.2.3 Configuración del Dispositivo Virtual de Android, AVD

El *AVD Manager* es una interfaz de usuario fácil de manejar que permite gestionar las distintas configuraciones de Dispositivos Virtuales de Android, AVD (*Android Virtual Devices*). Un AVD es una configuración del emulador de Android que permite modelar diferentes características de dispositivos que utilizan dicho sistema operativo.

Se pueden crear tantos AVDs como se quieran, de modo que se puedan realizar pruebas de las aplicaciones sobre dispositivos (virtuales) con distintas configuraciones.

En la ventana (***Android SDK and AVD Manager***), desde el apartado ***Virtual Devices*** y desde ***New*** para crear uno nuevo. Aparecerá el diálogo ***Create new Android Virtual Device (AVD)***. Se debe indicar nombre, se elige la versión de Android que soportará y un tamaño de al menos 256 megas de almacenamiento a la tarjeta SD. El resto de opciones pueden quedarse por defecto, pueden verse más detalles en la web de configuración. (Android Developers, Hardware options, 2012)

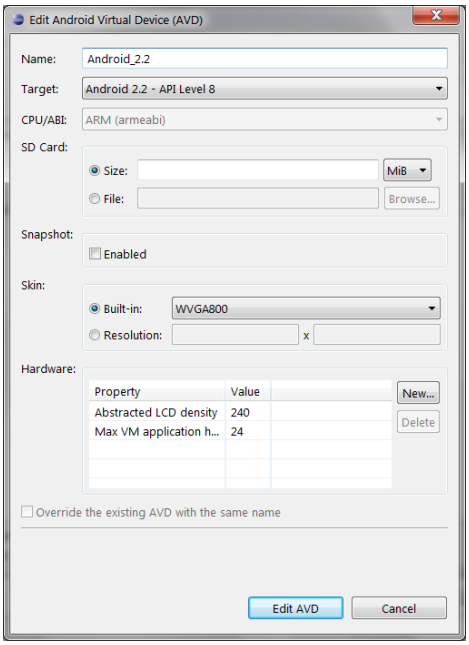


Ilustración 15. Creación de un dispositivo virtual

3.2.2.4 Creación de un proyecto Android

El plugin ADT proporciona la interfaz para crear proyectos Android de forma rápida y sencilla. Para crearlo:

1. Seleccionar ***File*** → ***New*** → ***Project***.
2. Seleccionar ***Android*** → ***Android*** Project y pinchar en ***Next.***
3. Seleccionar los contenidos del proyecto:
    - Introducir el nombre del proyecto. Éste será el nombre de la carpeta donde se cree el proyecto.
    - Debajo de ***Contents***, seleccionar ***Create new project in workspace***. Establecer la ubicación del proyecto.
    - Debajo de ***Target***, seleccionar un ***Android target***, que será utilizado como nivel de compilación del proyecto. Esto indica qué plataforma Android se utilizará.
    - En ***Properties***, rellenar los campos necesarios:
        - ***Application name***. Título legible de la aplicación. El nombre que aparecerá en el dispositivo del usuario.
        - ***Package name***. Espacio de nombres para la aplicación. Sigue la nomenclatura de los paquetes de java.
        - ***Min. sdk version***. Entero que indica el nivel mínimo de API requerido para que la aplicación se ejecute correctamente.
4. ***Finish***.

3.2.2.5 Compilación y ejecución de la aplicación

Eclipse, junto con el plugin ADT, proporcionan un entorno de desarrollo donde la mayor parte de los detalles del proceso de compilación son transparentes al desarrollador. Por defecto, el proceso de compilación se está ejecutando continuamente en segundo plano mientras que el hace cambios en el proyecto.

Cuando Eclipse compila automáticamente la aplicación, habilita la depuración y firma el *.apk* con una clave por defecto. Cuando se ejecuta la aplicación, Eclipse invoca ADB e instala la aplicación en un dispositivo o emulador, de manera que el desarrollador no tenga que hacer estas tareas manualmente. A continuación se explican los pasos para ejecutar una aplicación utilizando Eclipse, tanto en un emulador con en un dispositivo real.

Ejecución en un emulador

Para ejecutar la aplicación, seleccionar ***Run*** → ***Run*** (o ***Run*** → ***Debug***) desde el menú de Eclipse. El plugin ADT creará automáticamente una configuración de ejecución por defecto para el proyecto. Eclipse entonces realizará las siguientes tareas:

1. Compilar el proyecto (si ha habido cambios desde la última compilación).
2. Crear una configuración de ejecución por defecto (si no existe ya una para el proyecto).
3. Instalar e iniciar la aplicación en el emulador, basado en *Deployment Target* definido en la configuración de ejecución.

Si se ejecuta la aplicación en modo *Debug*, se iniciará en modo *Waiting for Debugger*. Una vez que el depurador se haya unido, Eclipse abrirá la perspectiva de depuración e iniciará la *Activity* principal de la aplicación. De otra forma, si se ejecuta la aplicación en modo normal, Eclipse instala la aplicación en el dispositivo e inicia la *Activity* principal de la aplicación.

Ejecución en un dispositivo

Antes de poder ejecutar la aplicación en un dispositivo, se deben realizar ciertas tareas de configuración para el mismo:

- Hay que asegurarse de que se puede depurar la aplicación estableciendo a true el atributo *android:debugable* de el elemento *<application>*.
- Habilitar *USB Debugging* en el dispositivo. En la mayoría de los dispositivos, esta configuración se encuentra en ***Settings*** → ***Applications*** → ***Development*** → ***USB Debugging***.

Una vez configurados estos parámetros y estando el dispositivo conectado vía USB,

instalar la aplicación en el dispositivo seleccionando ***Run*** → ***Run*** (o ***Run*** → ***Debug***) desde el menú de Eclipse.

3.2.2.6 Depuración de la aplicación

Si se desarrolla la aplicación utilizando Eclipse junto con el plug-in ADT, entonces se puede hacer uso también del depurador de JAVA junto con DDMS para depurar las aplicaciones.

DDMS (Dalvik Debug Monitor Server en inglés), es una herramienta incluida en Android y que proporciona utilidades al usuario tales como captura de pantalla, información de los hilos ejecutando en el dispositivo, información de trazas, procesos y estados, simulación de llamadas y SMS entrantes y muchas más cosas.

Para acceder al depurador de JAVA y a DDMS, Eclipse muestra ambas como perspectivas, que son vistas adaptadas para Eclipse donde se muestran diversas pestañas y ventanas dependiendo de en qué perspectiva se encuentre uno. Eclipse también se encarga de iniciar el proceso ADB automáticamente, de modo que el desarrollador no se tenga que preocupar por ello.

La perspectiva Debug en Eclipse

La perspectiva para la depuración en Eclipse, Debug, da acceso a las siguientes pestañas:

- ***Debug***: muestra las aplicaciones que han sido previamente depuradas o que se están depurando en este momento y los hilos que se están ejecutando actualmente.
- ***Variables***: cuando se establecen puntos de parada (breakpoints), muestra los valores de las variables durante la ejecución del código.
- ***Breakpoints***: muestra un listado con los puntos de parada que se han establecido en el código de la aplicación.
- ***LogCat***: permite ver mensajes de trazas del sistema en tiempo real. La pestaña LogCat también está disponible en la perspectiva DDMS.

Se puede acceder a esta perspectiva pinchando en ***Window*** → ***Open Perspective*** → ***Debug***.

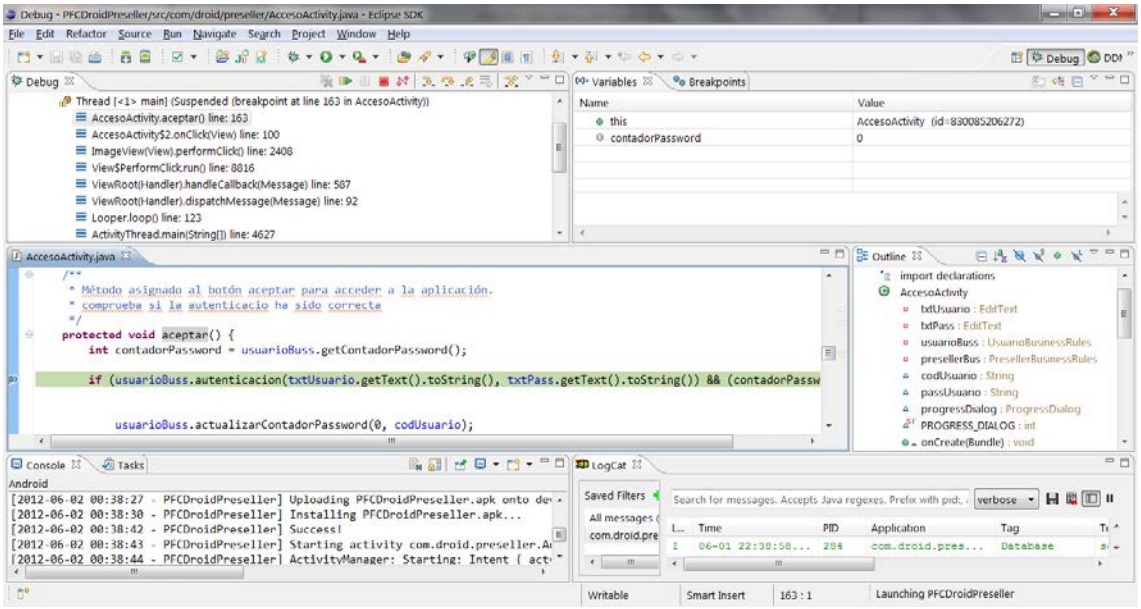


Ilustración 16. Perspectiva Debug de Eclipse

La perspectiva DDMS en Eclipse

Esta perspectiva de Eclipse permite al desarrollador acceder a todas las características de DDMS desde dentro del entorno de desarrollo Eclipse. Las secciones de DDMS que están disponibles:

- ***Devices***: muestra un listado de los dispositivos y ADVs que están conectados con ADB.
- ***Emulator Control***: permite llevar a cabo las funciones del dispositivo. LogCat: permite ver mensajes de trazas del sistema en tiempo real.
- ***Threads***: muestra los hilos que se están ejecutando actualmente dentro de la máquina virtual.
- ***Heap***: muestra el uso de la pila por parte de la máquina virtual.
- ***Allocation Tracker***: muestra la asignación de memoria de los objetos.
- ***File Explorer***: permite explorar el sistema de archivos del dispositivo.

Para acceder a la perspectiva DDMS, ***Window → Open Perspective → DDMS***. Si no aparece DDMS, entonces hacer ***Window → Open Perspective →* Other...** y seleccionar DDMS en la ventana que aparece.

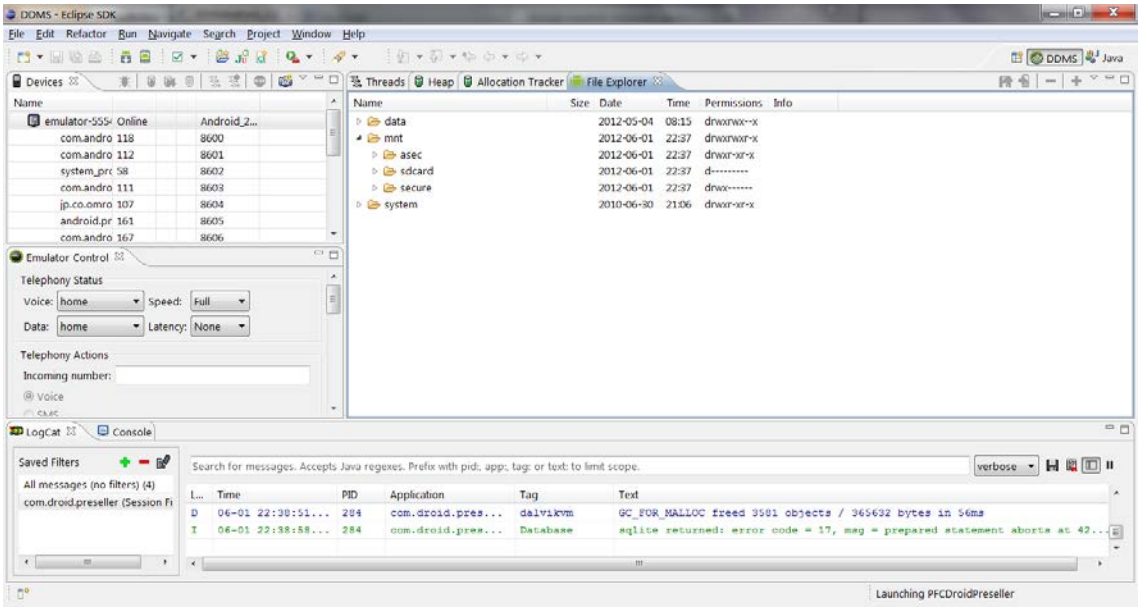


Ilustración 17. Perspectiva DDMS en Eclipse

3.2.3 Explorador SQlite

Buscando simplicidad y rapidez en el proceso, se determinó la utilización de una aplicación web, concretamente Sqlite Manager (Code.google.com, SQLite Manager, 2011).

Ilustración 18. Logotipo de sqlite-manager

SQLite Manager es un SQLite gestor de bases de datos gratuito como complemento para Firefox. Al ofrecer el software como una extensión de Firefox, SQLite Manager está disponible en diferentes plataformas y se caracteriza por no necesitar ningún servidor y su acceso directo al fichero de la base de datos. Utiliza la librería SQLite integrada en su aplicación de hospedaje (Firefox), y por lo tanto compatible con las versiones de los archivos de base de datos con el apoyo de la biblioteca de Firefox.

Permite al usuario ver el contenido de una o más bases de datos Sqlite, gestionar tablas, índices, vistas, ver y administrar los registros, construir una consulta SQL y ejecutarla, exportar datos a CSV o XML, visualizar el contenido de tablas existentes, modificar dicho contenido, etc.

Para la instalación de este software, en primer lugar, hay que descargar e instalar Firefox y, a continuación, en la pestaña Herramientas, seleccionar Complementos. Posteriormente, se busca, de entre los complementos existentes para Firefox, el complemento SQLite Manager y se instala.

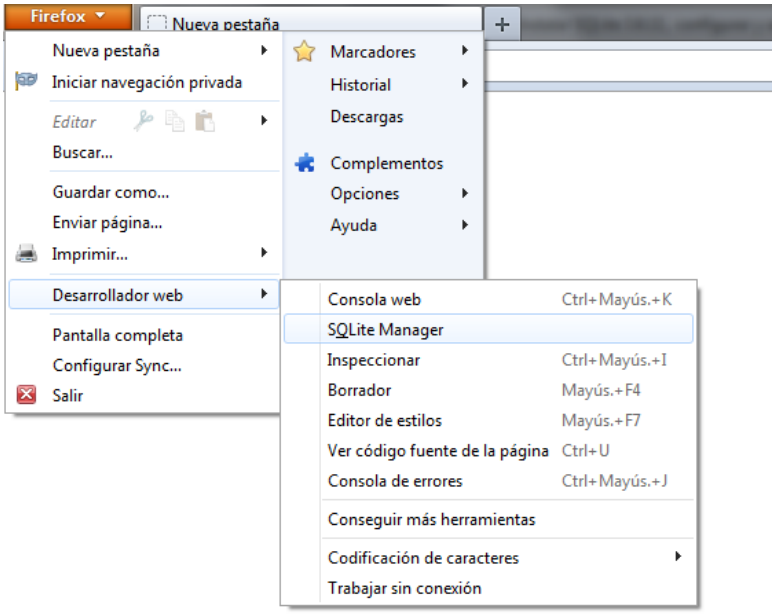


Ilustración 19. Acceso a SQLite Manager

3.2.4 IIS

El IIS (*Internet Information Services*) es un servidor web y un conjunto de servicios para el sistema operativo Microsoft Windows. Originalmente era parte del *Option Pack* para Windows NT. Luego fue integrado en otros sistemas operativos de Microsoft destinados a ofrecer servicios, como Windows 2000 o Windows Server 2003. Windows XP Profesional incluye una versión limitada de IIS. Los servicios que ofrece son: FTP, SMTP, NNTP y HTTP/HTTPS.

Ilustración 20. Logotipo del IIS7

Antiguamente se denominaba *PWS* (*Personal Web Server*), y actualmente forma parte de la distribución estándar de *Windows*, de modo que no se necesita una licencia extra para instalarlo. Este servicio convierte un PC en un servidor web para Internet o una intranet, es decir que en los equipos que tienen este servicio instalado se pueden publicar páginas web tanto local como remotamente.

Los servicios de *Internet Information Services* proporcionan las herramientas y funciones necesarias para administrar de forma sencilla un servidor web seguro.

El servidor web se basa en varios módulos que le dan capacidad para procesar distintos tipos de páginas. Por ejemplo, Microsoft incluye los de Active Server Pages (ASP) y ASP.NET. También pueden ser incluidos los de otros fabricantes, como PHP o Perl.

Capítulo IV. Proceso de desarrollo

Una vez asimilados todos los conceptos expuestos en los capítulos anteriores, es momento de pasar a la fase de desarrollo de la aplicación que ha sido planteada para este proyecto.

En primer lugar se definirá el escenario propuesto para llevar a cabo el proceso de desarrollo y se realizará un análisis de los requisitos de usuario de la aplicación. A continuación se abordará la fase de diseño del proyecto, se detallarán los aspectos más relevantes de la implementación de la aplicación, el manual de usuario y las pruebas realizadas. Y por último se hará un resumen del proyecto con su presupuesto justificado.

4.1 Escenario propuesto

A la hora de la toma de especificaciones se ha decido tomar como referencia una empresa de venta de bebidas. Dicha empresa se dedicaría a la venta de sus productos. La empresa contaría con diferentes tipos de trabajadores entre los que se encontrarían los preventas encargados de ir visitando a los diferentes clientes.

La aplicación que se va a implementar estaría pensada para estos trabajadores encargados de ir realizando las diferentes visitas, puedan ir tomando nota de los diferentes pedidos, puedan gestionar el préstamo de los equipos de frío en los que guardar los productos y otras necesidades de los clientes, para posteriormente comunicarlo a la central y que los pedidos puedan ser entregados.

Supuestamente esta empresa estaría tomando nota de los pedidos y demás datos manualmente, la propuesta por tanto es crearles un sistema móvil que les permita gestionar todas las acciones que se realizan, que sea de fácil uso y que permita el envío de los datos capturados de forma automática.

4.2 Fase de análisis

En este apartado se especificarán los requisitos de usuario que se han tenido en cuenta para el desarrollo del proyecto.

4.2.1 Especificación de requisitos de usuario.

El objetivo del proyecto ha sido desde un principio el estudio y la implementación de una aplicación Android aplicable a la vida empresarial actual. Es por ello que se ha decidido implementar una aplicación que cumpliese con las supuestas necesidades de una empresa cuyo negocio se basase la venta de productos a través de usuarios preventa

o comerciales. Estos preventas han de ser los encargados de visitar a su cartera de clientes, y una vez en el cliente poder gestionar los pedidos correspondientes.

De esta forma el objetivo principal de la aplicación ha de ser el facilitar y automatizar la labor de los preventas de una empresa de venta de artículos. Es por ello, que la aplicación deberá ser capaz de gestionar las tareas o funciones que se han supuesto para que los usuarios realicen o desempeñen este trabajo.

Entre otras, las funcionalidades que la aplicación ha de ofrecer son:

- Visualización del listado con los productos disponibles para la venta.
- Visualización del detalle de cada producto.
- Consulta del listado de los clientes.
- Consulta y modificación del detalle de los clientes.
- Captura de los pedidos solicitados por el cliente.
- Anotación de posibles incidencias
- Aplicación descuentos y promociones sobre el pedido.
- Realización encuestas a los clientes.
- Visualización del histórico de los pedidos de cada cliente.
- Captura de comentarios asociados a los clientes.
- Gestión del préstamo y el mantenimiento de los equipos para el mantenimiento de los artículos, si fuera necesario.
- Captura de los códigos de barras o códigos QR de los equipos prestados.
- Captura de fotos.
- Geo-localización de los clientes.
- Visualización de informes diarios para tener una visión de la productividad del día laboral.
- Envío y recepción de los datos capturados en el dispositivo.

4.2.2 Casos de uso

A continuación se detallan los casos de uso de la aplicación, se ofrecen los objetivos de los mismos, los actores que están involucrados en él (en este caso siempre se tratará del usuario del dispositivo), las precondiciones (estado del sistema que se tiene que cumplir para que el caso de uso se pueda instanciar), las pos condiciones (estado del sistema una vez instanciado el caso de uso) y el escenario básico (pasos principales del caso de uso ordenados).

Se definen tres bloques de casos de uso: los casos de uso de acceso a la aplicación, los casos de uso sobre los clientes y los casos de uso del menú principal.

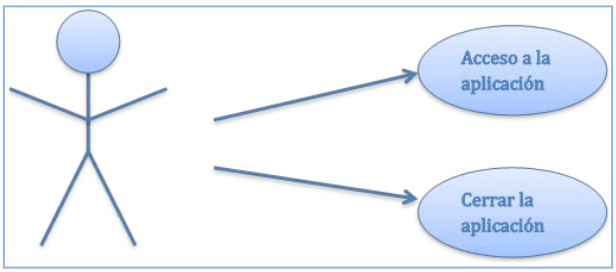


Ilustración 21. Caso de uso de acceso

CU-01: Acceso a la aplicación			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Autenticación del usuario en la aplicación.			
Escenario básico:	1. Introducir el usuario o verificar el que se ha auto rellenado (si solo hay un usuario en la tabla *user* se auto rellanará por defecto). 2. Introducir la contraseña. 3. Autenticar al usuario en caso de que el usuario y la contraseña sean correctos. 4. Si la contraseña se introduce mal diez veces consecutivas la aplicación se bloqueará.		
Precondiciones:	Tener datos en la tabla *user*.		
Poscondiciones:	Una vez autenticado se actualizará la versión de la aplicación en la Base de Datos en caso de que haya cambiado		

Tabla 1. Caso de uso CU-01

CU-02: Cerrar la aplicación			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Cerrar la aplicación			
Escenario básico:	1. Desde la pantalla de acceso se podrá cerrar la aplicación.		
Precondiciones:	----		
Poscondiciones:	Cada vez que se cierre la aplicación la se realizará una copia de la Base de Datos en el directorio de *backup* la tarjeta de memoria como copia de seguridad.		

Tabla 2. Caso de uso CU-02

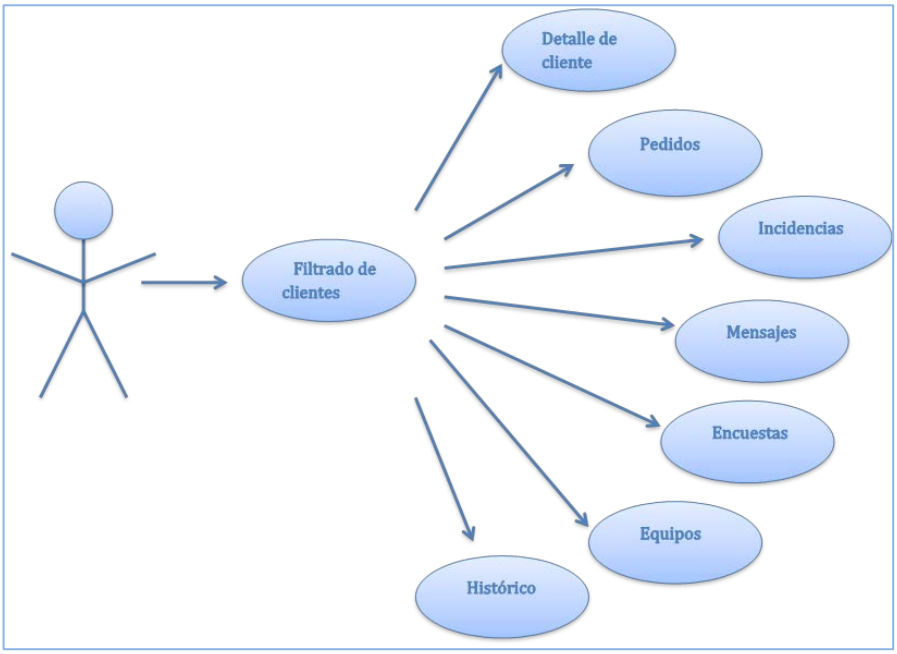


Ilustración 22. Casos de uso sobre los clientes

CU-03: Filtrado de clientes			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Realización de búsquedas de los clientes			
Escenario básico:	1. Desde la pantalla principal de la aplicación se podrán buscar clientes: • Cuya visita este planeada para el día de hoy. • Buscar clientes por rutas. • Buscar clientes por código o por nombre. 2. Una vez filtrados los clientes se podrá seleccionar el cliente deseado y realizar sobre él cualquiera de las posibles opciones.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 3. Caso de uso CU-03

CU-04: Visualización o modificación del detalle del cliente			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Visualización o modificación del detalle del cliente			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Datos*. 3. Aparecerá el detalle del cliente y se podrá por las cuatro ventanas que lo componen. 4. Desde el botón de menú se podrá seleccionar la opción de modificar. 4.1. Esta opción hará que todos los campos se vuelvan editables a excepción del código del cliente. 4.2. Una vez modificados los datos, desde el menú se podrá seleccionar la opción de guardar, para almacenar las modificaciones en la Base de Datos. 4.3. Si se sale de esta opción si haber guardado los cambios, el sistema deberá avisar de que se perderán las modificaciones no guardadas.		
Precondiciones:	Tener datos en la tabla *clientes*.		
Poscondiciones:	Una vez autenticado se actualizará la versión de la aplicación en la Base de Datos en caso de que haya cambiado		

Tabla 4. Caso de uso CU-04

CU-05: Captura, modificación o eliminación de pedidos			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán dar de alta pedidos nuevos asociados a un cliente o modificar o eliminar los existentes.			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Pedidos*. 3. Aparecerá el listado con los pedidos asociados al cliente en caso de que ya haya alguno. 4. Desde el botón de *más* se podrá añadir un nuevo pedido. 5. Una vez tomada nota de un nuevo pedido, pasará a aparecer en el listado. 6. Los pedidos podrán aparecer con dos estados diferentes confirmados y sin confirmar. 7. Solo los pedidos sin confirmar podrán modificarse. 8. Sin embargo podrán borrarse los pedidos confirmados y sin confirmar.		

Precondiciones:	----
Poscondiciones:	Una vez realizado un pedido sobre un cliente en el listado principal de la aplicación aparecerá marcado en verde la etiqueta *Venta efectuada* para indicar que ya ha sido visitado.

Tabla 5. Caso de uso CU-05

CU-06: Captura de una incidencia			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
El caso de visitar un cliente y que éste no haga ningún tipo de pedido, se deberá tomar nota de cual ha sido la razón.			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Incidencia*. 3. Aparecerá el listado con todos los posibles casos de incidencias registrados. 4. Se deberá seleccionar la razón dentro del listado.		
Precondiciones:	----		
Poscondiciones:	Una vez capturada la incidencia en el listado principal de la aplicación aparecerá marcado en rojo la razón por la que ese cliente no ha realizado ningún pedido.		

Tabla 6. Caso de uso CU-06

CU-07: Recoger un mensaje asociado a un cliente			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrá recoger un mensaje a comentario asociado al cliente			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Comentarios*. 3. Aparecerá el listado con las posibles cabeceras que se pueden poner al comentario. 4. Se debe seleccionar una de ellas y aparecerá la pantalla en la que introducir el comentario, o en caso de que ya haya un comentario guardado para ese cliente y bajo esa cabecera podrá modificarse. 5. Si se sale de esta opción si haber guardado los cambios, el sistema deberá avisar de que se perderán las modificaciones no guardadas.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 7. Caso de uso CU-07

CU-08: Realización encuestas al cliente			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán realizar diferentes encuestas predefinidas al cliente seleccionado.			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Encuestas*. 3. Aparecerá el listado con las posibles encuestas que se puedan realizar al cliente. 4. Se debe seleccionar una de ellas y aparecerá la secuencia de pantallas de las preguntas asociadas a esta encuesta y se deberá ir seleccionando una de las posibles respuestas para cada pregunta.		
Precondiciones:	Tener las encuestas definidas en las tablas *encuestas* y *preguntas.*		
Poscondiciones:	----		

Tabla 8. Caso de uso CU-08

CU-9: Captura, modificación o eliminación de equipos de frío			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán dar de alta el préstamo de nuevos equipos de frío o modificar o dar de baja los existentes.			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Equipos de Frío*. 3. Aparecerá el listado con los equipos asociados al cliente en caso de que ya haya alguno. 4. Desde el botón de *más* se podrá añadir un nuevo equipo. 5. Una vez tomada nota de un nuevo equipo, pasará a aparecer en el listado. 6. La captura del detalle del equipo, permitirá chequear si esta funcionando correctamente, si los productos que tiene es un interior no son de la competencia, se podrán tomar notas, fotos y capturar el código QR o código de barras para confirmar su autenticidad.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 9. Caso de uso CU-09

CU-10: Visualización del histórico de pedidos del cliente			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán visualizar los seis últimos pedidos asociados a cada cliente.			
Escenario básico:	1. Seleccionar un cliente sobre la lista. 2. Cuando aparezca el menú emergente se ha de seleccionar la opción de *Histórico*. 3. Aparecerá el listado con los últimos pedidos asociados al cliente ordenados por fecha. Solo se guardarán los seis últimos pedidos de cada cliente. 4. Pulsando sobre cada uno de ellos se podrá ver el detalle del pedido.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 10. Caso de uso CU-10

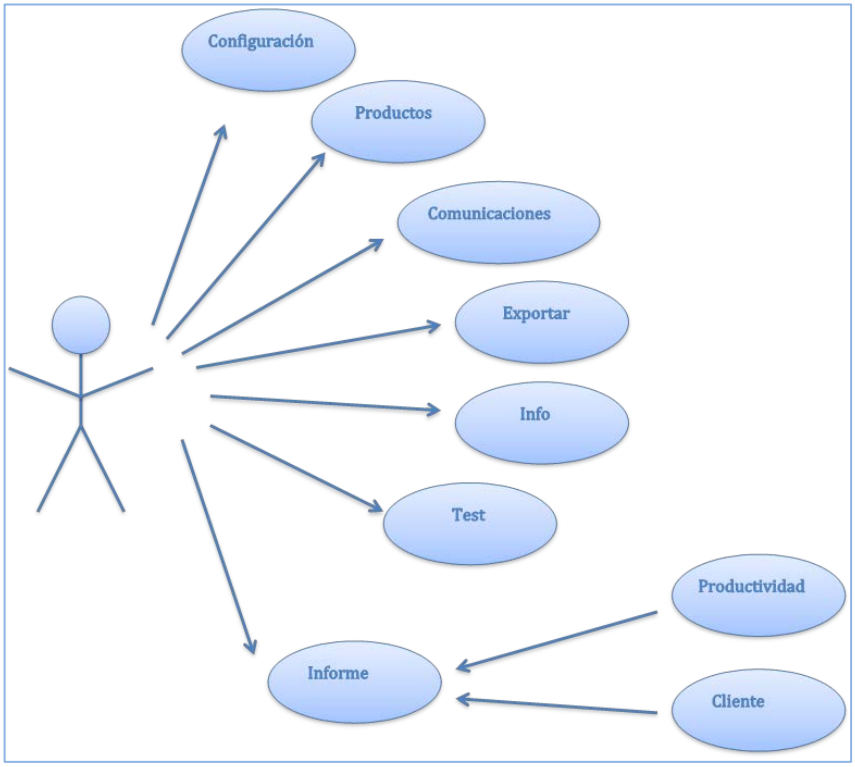


Ilustración 23. Casos de uso del menú principal

CU-11: Configuración de aplicación			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán configurar algunos de los parámetros de la aplicación.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú y se deberá pulsar sobre la opción *Configuración*. 2. Para acceder a la configuración habrá que introducir la contraseña almacenada en la tabla *parameters*. 3. Una vez autenticado aparecerá la pantalla de configuración en la que se podrán modificar algunos parámetros de la aplicación. Estos parámetros serán las rutas donde se guardaran los ficheros, los nombres de los ficheros, la dirección del servidor de aplicaciones, el identificador del dispositivo.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 11. Caso de uso CU-11

CU-12: Visualización del listado y detalle de los productos			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán visualizar los productos definidos en la aplicación.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú principal y se deberá pulsar sobre la opción *Productos*. 2. Aparecerá el listado con todos los productos y se podrán realizar búsquedas sobre ellos, para poder acotar las búsquedas. 3. Pulsando sobre un producto de la lista se podrá acceder al detalle del mismo. Esta información será solo de consulta.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 12. Caso de uso CU-12

CU-13: Exportación de los datos			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán exportar los datos modificados o creados nuevos en la Base de Datos a un fichero de texto externo.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú y se deberá pulsar sobre la opción *Exportar*. 2. Todos aquellos datos marcados en la Base de Datos como nuevos o modificados se escribirán en un fichero, en la ruta que se haya indicado en la configuración.		
Precondiciones:	----		
Poscondiciones:	Se generará un fichero con las modificaciones de la base de datos. Este fichero se llamará y se almacenará tal y como se hayan definido los parámetros de configuración.		

Tabla 13. Caso de uso CU-13

CU-14: Proceso de comunicaciones de la aplicación			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
El proceso de comunicación se encargará de exportar los datos, enviarlos al servidor, bajarse datos en caso de que los haya, importarlos en la Base de Datos y actualizar los estados para no volver a exportar los datos correctamente enviados.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú y se deberá pulsar sobre la opción *Comunicaciones*. 2. EL primer proceso que se llevará a cabo será la exportación de todos aquellos datos que hayan sido insertado nuevo o modificados en la Base de Datos a un fichero de texto. 3. Se establecerá una conexión con el servidor configurado en los parámetros de la aplicación, y se subirá el fichero al servidor. 4. Una vez subido el fichero exportado se comprueba si hay algún fichero que descargar, de la ruta previamente configurada en los parámetros, en caso de que haya un fichero que descargar se bajará al teléfono. 5. En caso que se haya bajado algún fichero de importación, se realizará el proceso de importación en la Base de Datos. 6. Una vez finalizado el proceso de comunicación se procederá a actualizar todos los registros de la Base de Datos para indicar que ya han sido enviados y no deben		

	enviarse en la siguiente comunicación. 7. Se deberá hacer un tratamiento de errores de todo el proceso teniendo en cuenta que se podría producir un fallo en cualquier punto de la comunicación.
Precondiciones:	Deberá haber un servidor de aplicaciones esperando las peticiones pertinentes.
Poscondiciones:	----

Tabla 14. Caso de uso CU-14

CU-15: Visualización de la información de la aplicación			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se ofrecerá una pantalla al usuario en la que se muestre la versión de la aplicación y aquella información que pueda ser de utilidad.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú y se deberá pulsar sobre la opción *Info*. 2. Aparecerá una pantalla con la información de la aplicación.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 15. Caso de uso CU-15

CU-16: Testeo del servidor			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrá testear la conexión del dispositivo y realizar una prueba de conexión con el servidor.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú y se deberá pulsar sobre la opción *Test*. 2. Se realizará un testeo de la comunicación con el servidor de aplicaciones para saber si la conexión tanto del dispositivo como del servidor están funcionando correctamente.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 16. Caso de uso CU-16

CU-17: Visualización de los informes de ventas			
Prioridad	**Estabilidad**	**Actores**	**Versión**
Alta	Modificable	Usuario	1.0
Objetivo:			
Se podrán visualizar los informes de ventas tanto por cliente como de productividad del día actual.			
Escenario básico:	1. Desde la pantalla principal de la aplicación, se podrá acceder al menú y se deberá pulsar sobre la opción *Informes*. 2. Aparecerá un menú con las dos posibles tipo de informe que se pueden mostrar. Informe por clientes o informe de productividad. 3. El informe por clientes, mostrará las ventas realizadas en el día actual asociadas a cada cliente, así como le relación de visitas realizadas frente a visitas planeadas y el importe total vendido. Pulsando sobre cada uno se podrá acceder al informe detallado por cliente. 4. El informe de productividad, mostrará la relación de visitas productivas y el porcentaje total de productividad.		
Precondiciones:	----		
Poscondiciones:	----		

Tabla 17. Caso de uso CU-17

Una vez tomados los requisitos de la aplicación, se ha procedido a la realización de la misma haciendo uso de los componentes Android y de JAVA necesarios para su implementación.

4.3 Fase de diseño

Durante esta sección se discutirán los aspectos relacionados con la fase de diseño dentro de ciclo de desarrollo de software. Concretamente se van a comentar los detalles relacionados con la arquitectura de la aplicación, el diseño de las clases y el diseño de la base de datos.

4.3.1 Arquitectura de la aplicación

La aplicación está estructurada en tres grandes módulos, que se corresponden con los tres paquetes principales de la aplicación.

Componente de interfaz de usuario: Es el componente encargado de implementar toda la parte gráfica de la aplicación. Para la implementación de la interfaz de usuario se ha hecho uso de *Activities* y de ficheros xml denominados *layouts*.

Componente de negocio: Es el principal de la aplicación. En él se engloba toda funcionalidad de negocio de la aplicación. Dentro de él se realizará la gestión de pedidos, productos, clientes, etc. Además, también gestionará las comunicaciones con el servidor.

Componente de acceso a base de datos: Como su propio nombre indica, es el encargado de gestionar la persistencia de datos. La aplicación hace uso de una base de datos SQLite3 además del sistema de ficheros del dispositivo.

Al crear un proyecto Android, en el *workspace* de Eclipse se genera una nueva carpeta con el nombre de dicho proyecto. Esta carpeta contiene una serie de subcarpetas y de ficheros que constituyen la anatomía completa de un proyecto Android.

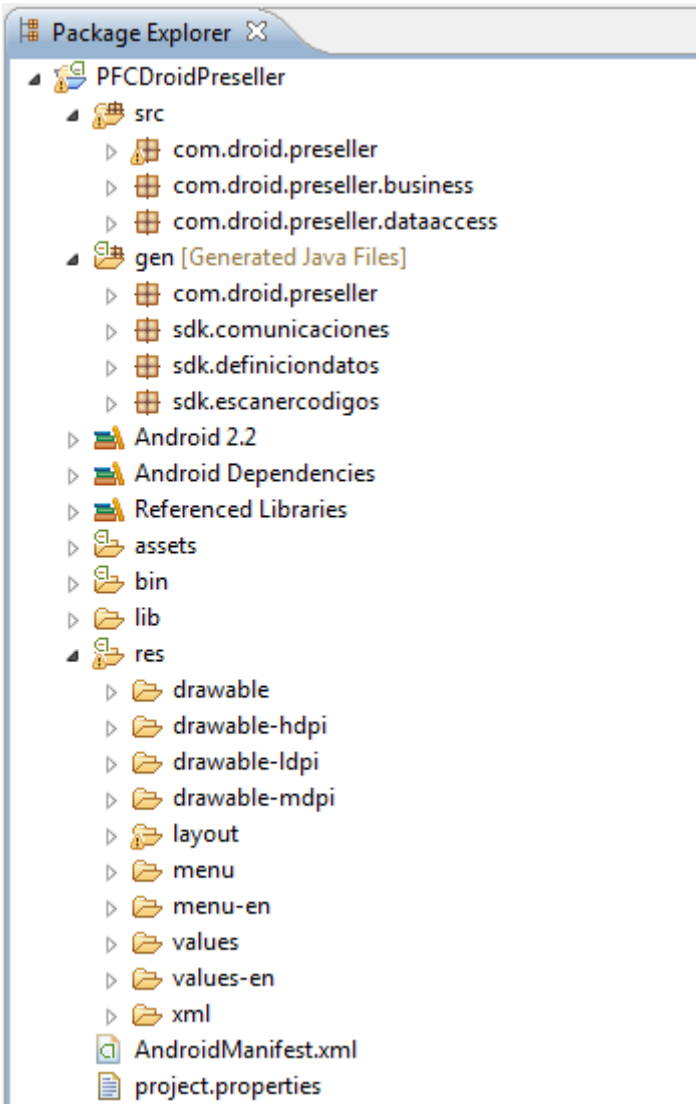


Ilustración 24. Estructura del proyecto Android

La carpeta **\src** es la carpeta que contiene los archivos fuente .java del proyecto. Utiliza la misma jerarquía de carpetas que la indicada por el nombre del paquete que se haya asignado. Se ha decidido seguir la estructura de tres paquetes diferenciados de la siguiente forma:

- ***com.droid.preseller*** paquete donde se almacenarán las *Activities*.
- ***com.droid.preseller.business*** paquete donde se almacenarán todas las clases del negocio de la aplicación.
- ***com.droid.preseller.dataaccess*** paquete bajo el que se encontrarán todas las clases de acceso a Base de Datos.

El código de negocio relacionado con las comunicaciones, el tratamiento de datos (necesario para la importación y la exportación) y el escaneo de códigos por razones de diseño se ha decidido implementarlo en tres proyectos de tipo librería de Android para facilitar su reutilización si fuese necesaria, estos proyectos son *sdk_comunicaciones*, *sdk_definiciondatos* y *sdk_escanercodigos*.

La carpeta **\res** alberga los recursos utilizados en el proyecto. Por recurso se entiende cualquier fichero externo que contenga datos o descripciones referentes a la aplicación, y que debe ser compilado junto a los ficheros fuente. Esta compilación permite al recurso ser accedido de forma más rápida y eficiente. Cada uno de los recursos utilizados en una aplicación para Android ha de estar localizado en la carpeta adecuada, en función de su naturaleza.

- **\drawable:** esta carpeta contendrá recursos para ser dibujados en la pantalla. Por ejemplo, imágenes con formato JPG, PING o GIF.
- **\layout:** esta carpeta contendrá *layouts* o diseños que construyen los interfaces. Pueden representar una la pantalla completa o simplemente una parte de ella como pueden ser una línea de una lista.
- **\values** y **\menus:** estas carpetas contendrá ficheros XML que declaran valores de diferentes tipos. Por ejemplo, cadenas de texto, menús, colores predefinidos, arrays de elementos, dimensiones, estilos, etc. Por convención, existirá dentro de esta carpeta un fichero XML por cada tipo distinto de recurso que se declare: ***strings.xml*** para los *strings*, ***colors.xml*** para los colores, ***ids.xml*** para los identificadores, etc.
- **\xml:** donde se ubicarán aquellos ficheros XML genéricos que pueden ser procesados como tales desde el código (es decir, utilizando un parseador de XML) como son los ficheros de preferencias.

Y la carpeta **\assets** contendrá la Base de Datos de la aplicación.

4.3.2 Diagrama de clases

Las clases utilizadas para resolver un determinado problema, sus atributos y métodos, la visibilidad que de estos tienen las demás clases, así como las relaciones que existen entre ellas y sus colaboradores, constituyen el modelo de clases. Mediante este tipo de modelos se expresa con mayor o menor nivel de detalle, la futura implementación del sistema y permite dar una idea bastante cercana de la forma en la que se ha abordado el problema.

En este apartado se entrará en los detalles de los componentes arquitectónicos de la aplicación. Esto incluye diagramas de clases para facilitar la comprensión de la estructura del proyecto. En cualquier caso se ha mantenido cierto nivel de abstracción dado que exponer la aplicación con todo detalle ocuparía cientos de páginas y no resultaría de utilidad.

En el siguiente diagrama de componentes se puede ver la secuencia de llamadas que se realizarían en un flujo normal de la aplicación. Como ya se definió anteriormente las *Activities* junto con los *layouts* se encargarán de implementar el aspecto visual de la aplicación, las clase de negocio de la aplicación se encargan de implementar toda la lógica y la funcionalidad, y por último, las clases de acceso a Base de Datos serán las encargadas de recuperar y actualizar los datos de la aplicación.

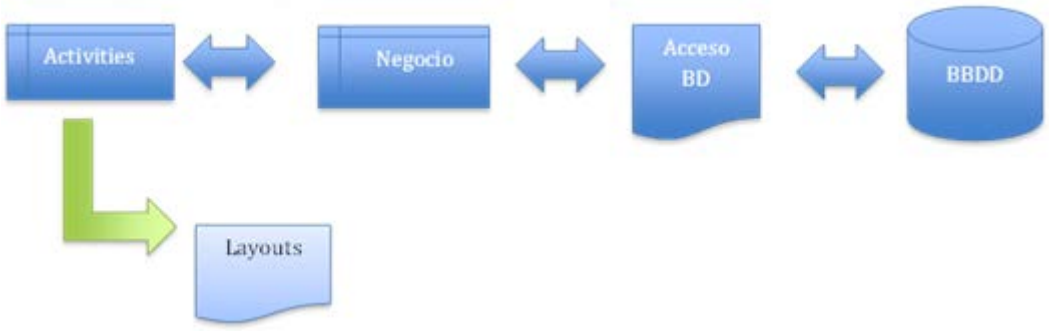


Ilustración 25. Diagrama de componentes

Cada una de las tablas que forman la Base de Datos se ha plasmado en el código a través de una clase que la represente. De esta forma se ha facilitado el acceso a los datos. Todas estas clases implementan la clase *PresellerAccesoDatosGeneralDB* de forma que todo el código de acceso a Base de Datos se concentra en único código. En la **Ilustración 26** puede verse la estructura de las clases asociadas a las tablas que componen la aplicación. La clase *PresellerAccesoDatosGeneralDB* será la única que implemente todos los métodos de acceso a la Base de Datos y el resto de clases asociadas a las tablas además de implementar ésta, tendrán definidas sus diferentes estructuras de columnas para personalizarlas.

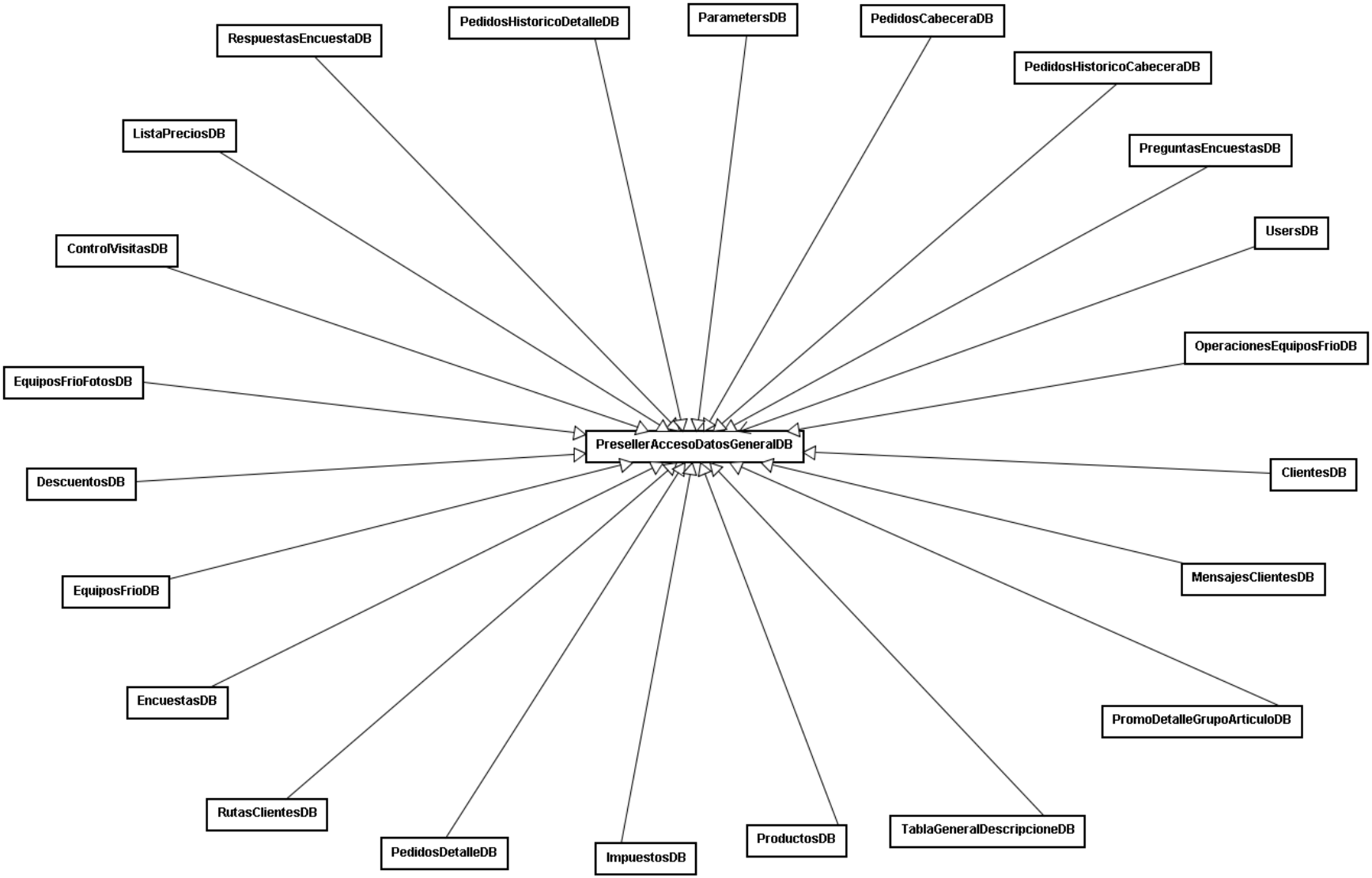


Ilustración 26. Diagrama de las clases asociadas a las tablas de la BD

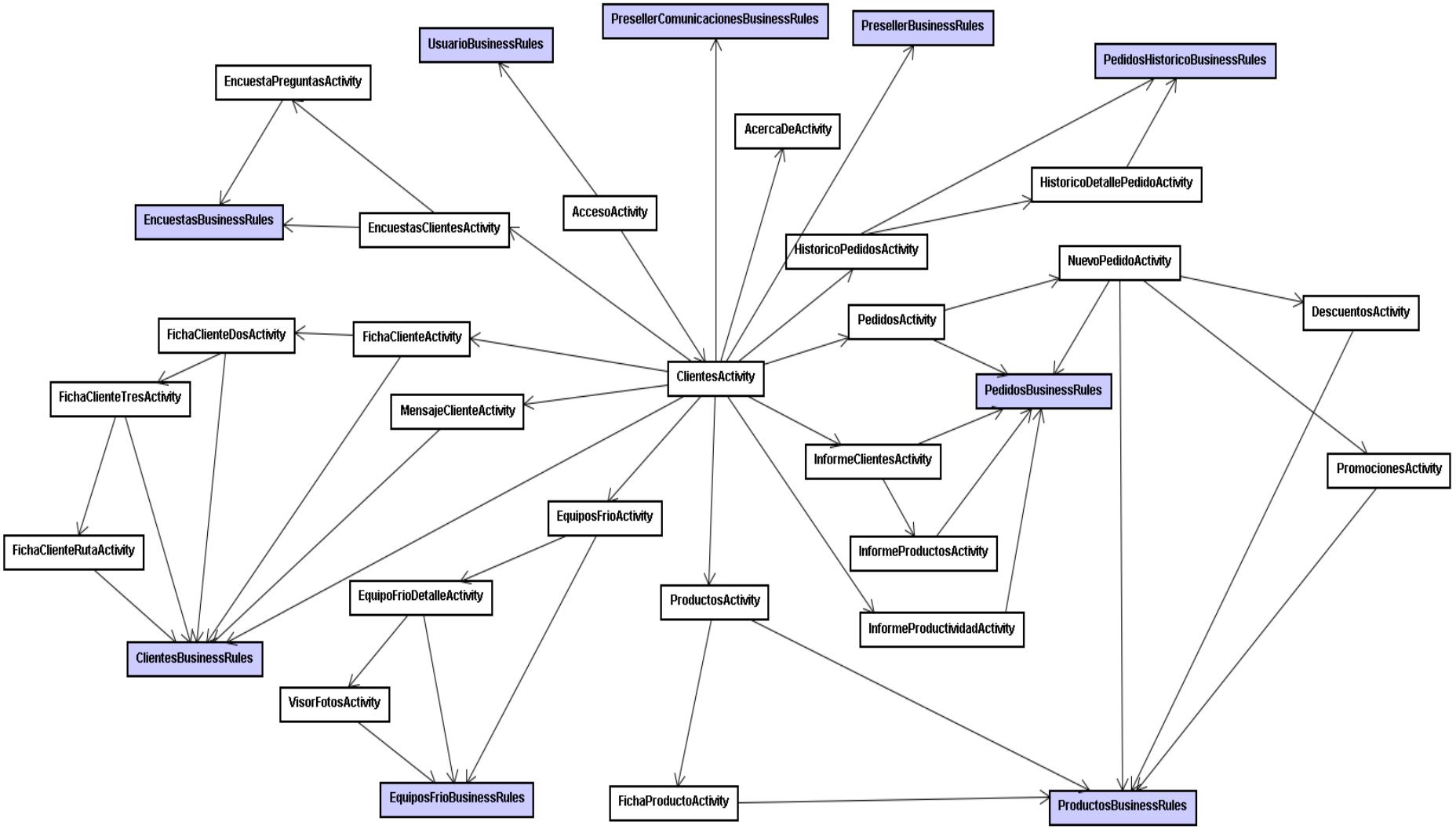


Ilustración 27. Diagrama de clases de las activities y clases de negocio de la aplicación

Se podría decir que la *Activity* principal de la aplicación es *ClientesActivity* ya que desde esta pantalla es desde la que se puede acceder a casi toda la funcionalidad de la aplicación. Por una parte se accede al menú principal de la aplicación y por otra al menú emergente que muestra las acciones posibles que se pueden realizar sobre un cliente seleccionado. En la **Ilustración 27** se puede ver la estructura de llamadas que se podrían hacer de unas *Activities* a otras. Las clases de negocio de la aplicación se han representado en el diagrama en color azul para diferenciarlas de las *Activities*. Estas clases de negocio se encargarán de obtener y tratar los datos antes de pasárselos a las activities.

Se ha intentado dividir el negocio en los grandes bloques de la aplicación para poder organizarlo de una forma coherente. Las clases de negocio son las siguientes: *ClientesBusinessRules, EncuestasBusinessRules, EquiposFrioBusinessRules, Globales, GlobalesPreseller, PedidosBusinessRules, PedidosHistoricoBusinessRules, PresellerBusinessRules, ProductosBusinessRules, UsuarioBusinessRules y PresellerComunicacionesBusinessRules.*

Para entrar un poco más en detalle a continuación se ofrece una breve explicación de algunas de las clases expuestas en los diagramas anteriores.

AccesoActivity: clase que extiende de *android.app.Activity* y se encarga de mostrar al usuario la pantalla de autenticación en la aplicación. Esta *Activity* estaría asociada al layout ***access.xml*** en el que se define los componentes visuales que se mostrarán al usuario. Esta pantalla ofrecerá un formulario de autenticación donde introducir usuario y contraseña y una vez se pulse el botón de aceptar se procesará la información introducida para comprobar si la autenticación es correcta.

UsuarioBusinessRules: esta clase se encarga de englobar todo el negocio relacionado con el usuario que esta utilizando la aplicación. Dispone de una serie de métodos como el encargado de verificar si el usuario y contraseña introducidos coinciden con los almacenados en la base de datos, o el que se encarga de ir aumentando el contador en caso de autenticaciones erróneas.

UsersBD: esta clase extiende *PresellerAccesoDatosGeneralDB* y se encarga de representar la tabla ***users*** de la Base de Datos. Tiene definido un *array* de *Strings* con todos los nombre de columnas que forman la tabla, y una constate asociada a cada nombre para poder referenciarlas en el código de la aplicación. Al implementar la clase *PresellerAccesoDatosGeneralDB* puede hacer uso de todos los métodos que componen esta clase. Todas las clases asociadas a cada una de las tablas seguirán la misma estructura, es por ello que no se considera necesario el detallar cada una de ellas en este apartado.

PresellerAccesoDatosGeneralDB: esta clase extiende de *android.database.sqlite.SQLiteOpenHelper* y se encarga de implementar todos los métodos relacionados con el acceso a la Base de Datos. Esta clase tendrá definidos dos atributos uno de tipo *SQLiteDatabase* que hará referencia al objeto que representa la

Base de Datos y otro de tipo *String* que se inicializará con el nombre de la tabla a la que se haga referencia en ese momento. En esta clase se implementarán los métodos encargados de realizar inserciones, búsquedas, actualizaciones o borrado de registros, así como los métodos que facilitan la creación de las *querys* necesarias.

ClientesActivity: se puede considerar que esta clase que extiende de *android.app.Activity* es la clase principal de la aplicación ya que desde ella se puede acceder a la mayoría de la funcionalidad de la misma. Esta *Activity* esta relacionada con el layout ***clientes.xml*** y ofrece al usuario un listado con los clientes (asociado a su vez al layout ***presellerclientlis.xml***) y un panel con el que poder filtrar y hacer búsquedas sobre ellos. Sobre el listado de clientes se podrá interactuar pudiendo realizar diferentes acciones sobre el cliente seleccionado y a su vez se ofrecerá un menú principal con las funciones generales de la aplicación.

ClientesBussinesRules: esta clase engloba la parte del negocio asociado con los clientes. Implementará los métodos encargados de mostrar y modificar los clientes, así como y cargar los diferentes filtros de búsqueda.

FichaClienteActivity, ***FichaClienteDosActivity***, ***FichaClienteTresActivity*** y ***FichaClienteRutaActivity*** son las clases que extienden *de android.app.Activity* encargadas de mostrar el detalle de los clientes.

PedidosActivity: clase que extiende de *android.app.Activity* encargada de mostrar un listado con los diferentes pedidos que puede tener un cliente asociados. También ofrecerá una opción para crear pedidos nuevos. Esta *Activity* está asociada al layout ***pedidos.xml*** y a su vez a ***presellerpedidoslist.xml*** que contiene la lista de pedidos.

NuevoPedidoActivity: esta clase que extiende *android.app.Activity* será la encargada de mostrar el detalle de los pedidos ya realizados y de permitir crear pedidos nuevos. Tendrá un menú para poder acceder a las *Activities* ***DescuentosActivity*** y ***PromocionesActivity*** ambas pantallas de visualización que ofrecen al usuario un resumen con todas las promociones y descuentos que se pueden aplicar al pedido.

PedidosBusinessRules: se podría decir que esta será una de las clases de negocio principales ya que englobará gran parte de la funcionalidad de la aplicación. Esta clase implementará los métodos encargados de calcular las promociones, descuentos, total de los pedidos, aplicar los impuestos pertinentes, así como las tareas de administración de pedidos para poder modificarlos o borrarlos.

ProductosBusinessRules: junto con ***PedidosBusinessRules*** se encargará de implementar el negocio necesario para la realización de pedidos. En esta clase de realizarán los cálculos para la obtención de los diferentes productos de regalos dependiendo de las promociones, además de obtener los diferentes precios e impuestos de los productos entre otras funcionalidades.

EquiposFrioActivity: clase que extiende de *android.app.Activity* encargada de mostrar un listado con los diferentes equipos de frío que puede tener un cliente asociados. También ofrecerá una opción para dar de alta nuevos equipos. Esta *Activity*

está asociada al layout ***pedidos.xml*** al igual que pedidos y a su vez a ***presellerequiposlist.xml*** que contiene la lista de los equipos. Esta clase también implementará *android.location.LocationListener* lo que permitirá capturar las coordenadas en este punto.

EquiposFrioDetalleActivity: esta clase que también extiende de *android.app.Activity* será la encargada de mostrar el detalle de los equipos a la hora de darlos de alta o de verificar que el equipo prestado está en las condiciones adecuadas. Desde esta Activity se permitirá tirar fotos que se podrán visualizar desde ***VisorFotosActivity***, así como hacer la llamada a la clase ***CameraCaptureActivity*** que permitirá capturar códigos de barras y códigos QR.

EquiposFrioBusinessRules: clase de negocio asociada a los equipos, encargada de guardar los datos capturados y mostrar los ya almacenados en la Base de Datos.

EncuetasClientesActivity: clase que extiende de *android.app.Activity* encargada de mostrar un listado con las diferentes encuestas que puede tener un cliente asociadas. Esta relacionada con el layout ***encuestas.xml*** y a su vez con el layout ***presellerencuentalist.xml*** encargado de mostrar los datos de la lista

EncuestaPreguntasActivity: esta clase que también extiende de *android.app.Activity* es la encargada de mostrar el detalle de cada encuesta mostrando cada una de las preguntas predefinidas en la tabla ***preguntas***. Según se vayan seleccionando las repuestas se almacenarán en la tabla ***respuestas***.

EncuestaBusinessRules: esta clase engloba todo el negocio relacionado con las encuestas, desde los métodos para obtener el listado de las encuestas asociadas a cada cliente, como los métodos necesarios para almacenar los datos capturados en la Base de Datos.

HistoricoPedidoActivity: clase que extiende de *android.app.Activity* encargada de mostrar un listado con el histórico de pedidos de cada cliente, este histórico esta compuesto como máximo por los seis últimos pedidos llevados a cabo para el cliente seleccionado. Esta relacionada con el layout ***historico.xml*** y a su vez con el layout ***presellerpedidoshistoricolist.xml*** encargado de mostrar los datos de la lista

HistoricoDetallePedidoActivity: clase que extiende de *android.app.Activity* y que se encarga de mostrar el detalle de los pedidos almacenados en el histórico.

PedidosHistoricosBusinessRules: esta clase englobará todo el negocio relacionado con la gestión de los históricos de la aplicación.

InformeClientesActivity: esta clase que extiende de *android.app.Activity* asociada al layout ***informeclients.xml*** y éste a su vez con ***presellerInformeclientelist.xml***. En esta pantalla se mostrará un resumen de las ventas realizadas en el día actual teniendo en cuenta los clientes que debían visitarse y lo que realmente se han visitado y el total de dinero neto vendido. Se ofrecerá una lista con todos aquellos clientes en los que se han realizado ventas en el día actual, y pulsando sobre los clientes se llamará a clase

InformeProductosActivity que ofrece un resumen de los productos vendidos a cada cliente.

InformeProductividad: esta clase mostrará el informe de productividad, extiende de *android.app.Activity* y estará asociada al layout ***informeproductividad.xml***.

PresellerPreferenceActivity: junto con las clases ***PresellerPreferenceActivity2*** y ***PresellerPreferenceActivity3*** formarán el conjunto de las preferencias de la aplicación y en ellas se configurarán los parámetros necesarios para el correcto funcionamiento. Estas clases extienden de *android.prefrerences.PreferenceActivity* y están asociadas a los layouts ***preferences.xml***, ***preferences2.xml*** y ***preferences3.xml*** en los que se configuran los valores por defecto de los parámetros.

PresellerComunicacionesBusinessRules: esta clase engloba el proceso de la comunicación de la aplicación. Esta clase extiende de ***PresellerAsyncTask*** que a su vez lo hace *android.os.AsyncTask.* Al extender de *AsyncTask* permitirá crear una tarea asíncrona que vaya actualizando el mensaje según vaya llevándose a cabo la comunicación.

ComunicacionesBusinessRules: en esta clase se implementarán los métodos de negocio relacionados con las comunicaciones.

TratamientoDatosBusiness: en esta clase se implementarán todos los métodos de tratamiento de datos para llevar a acabo los procesos de importación y exportación de la Base de Datos a fichero, y de ficheros a la Base de Datos.

4.3.3 Persistencia

Para la implementación de la capa de persistencia y de almacenamiento de la información requerida por la aplicación se ha utilizado una base de datos de SQLite en la que se almacena por un lado, la información necesaria para el funcionamiento de la aplicación, por otro la propia información recogida durante la utilización de la misma y finalmente, la información relativa a la autenticación del usuario.

Como se ha podido ver en la **Ilustración 26** la base de datos está compuesta por una serie de tablas diseñadas con la finalidad de poder albergar los datos necesarios para el desarrollo y funcionamiento de la aplicación.

Todas las tablas van a tener una columna *status_mobile* que va a permitir a la aplicación el poder gestionar cuando un registro se ha modificado o se ha creado nuevo y ha de procesarse para el envío. Un vez se produzca el proceso de comunicación y los datos e envíen las base de datos deberá actualizarse para marcar los datos que ya se han enviado.

Para llevar a cabo el proceso de gestión de datos va a haber dos tablas de gestión de la aplicación, denominadas *TDTablas* y *TMAEDiccicionarioDatos*. En la tabla *TDTablas* se van a indicar los nombres de las tablas del sistema y si se trata de tablas de importación, de exportación o de ambos procesos. Y la tabla *TMAEDiccicionarioDatos* va a contener la relación de las tablas, con sus campos y longitudes máximas de datos para poder ir generando las *querys* necesarias para la importación y la exportación de datos.

Por entrar un poco de detalle se podría decir por ejemplo que las tablas de productos son sólo de importación ya que los datos solo deben importarse en el dispositivo. Las tablas de pedidos serían tablas de exportación ya que los datos se recogen en el dispositivo, se enviarán al servidor y se vaciarán para continuar recogiendo nuevos pedidos. Las tablas de clientes son tablas tanto de importación como de exportación ya que los clientes son definidos por la empresa e importados en el dispositivos, pero podrían ser modificados desde la aplicación, por lo que las modificaciones deben enviarse.

4.3.4 Servidor de aplicaciones

Para poder llevar a acabo las pruebas de comunicaciones de la aplicación ha sido necesaria la configuración de un servidor de aplicaciones que sea capaz de recibir los ficheros subidos por los dispositivos y de ofrecer un soporte para publicar los ficheros que los dispositivos han de bajarse.

Se ha decidido implementar el servidor en una plataforma IIS, tendrá publicada una pagina *asp* con los diferentes métodos que serán invocados desde los dispositivos. Estos métodos serán básicamente:

- *pruebasp*. Método encargado de testear la conexión con el servidor. Devuelve la cadena OK ante una petición. Se utiliza para saber si la conexión está funcionando. La llamada que se haría sería de la siguiente forma *http://servidor/pfcserver/download.asp?accion=pruebasp&palmtop=00001* en el parámetro *accion* se indicará el nombre del método que se está invocando y en el parámetro *palmtop* se indicará el identificador del móvil que está comunicando, todas las llamadas a los diferentes métodos seguirán esta estructura.
- *getsize*. Método encargado de devolver el tamaño del fichero en caso de que el dispositivo, haya subido algún fichero previamente.
- *enviar*. Método encargado de guardar el fichero enviado por el dispositivo en la carpeta asociada al identificador del dispositivo y concatenarlo con el existente en caso de que haya un fichero subido previamente.

- Para la descarga de ficheros no hay necesidad de ningún método porque desde el código de la aplicación Android se accederá directamente a la url donde este publicados los ficheros para descargar y se bajarán al móvil.
- *eliminarhostpc*. Y un último método encargado de borrar el fichero una vez se haya descargado con éxito.

Todos estos métodos serán invocados desde el teléfono. La estructura del servidor estará formada por una directorio principal donde se publica la pagina *asp*, y una estructura de carpetas de forma que cada dispositivo pueda almacenar sus datos en un directorio.

En el *Anexo* de la memoria se puede ver un ejemplo del código utilizado para la recepción de los ficheros enviados por los dispositivos.

4.4 Fase de codificación

En esta sección se van a comentar los aspectos de la implementación más importantes y que han supuesto un mayor desafío o han resultado más llamativos o novedosos.

El objetivo de este capítulo es mostrar el código más relevante, de forma que se mostrarán los detalles los detalles más básicos para el desarrollo de l aplicación, algunas particularidades y casos especiales encontrados durante el desarrollo.

4.4.1 Android Manifest

Se va a comenzar explicando la configuración del fichero *manifest* de la aplicación, que como se explicó previamente es el fichero en el que se definen las propiedades, actividades, servicios y permisos que la componen. A continuación se detallarán algunos de los puntos más relevantes del fichero, en primer lugar se va a definir la cabecera del fichero xml, el paquete al que pertenece la aplicación y la versión del código.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    android:debuggable="true"
    package="com.droid.preseller"
    android:versionCode="1"
    android:versionName="1.0" >
```

Ejemplo 2. Cabecera del fichero *manifest*

La *Activity* principal de la aplicación será aquella que se llame en primer lugar una vez se arranque. En este proyecto la actividad principal es la denominada *AccesoActivity*. Se trata de la actividad desde la que el usuario deberá autenticarse introduciendo su código y contraseña para poder acceder. En el fichero *manifest* es necesario indicar cual será la actividad principal de la aplicación, mediante las sentencias *android.intent.action.MAIN* y *android.intent.action.LAUNCHER*.

```
<activity
       android:label="@string/app_name"
       android:name=".AccesoActivity"
       android:screenOrientation="nosensor"
       android:theme="@style/customTheme" >
      <intent-filter >
             <action android:name="android.intent.action.MAIN" />
             <category android:name="android.intent.category.LAUNCHER" />
       </intent-filter>
</activity>
```

Ejemplo 3. Activity principal de la aplicación

La aplicación hace uso de diferentes características del teléfono, para ello es necesario indicar en el fichero *manifest* cuales son los permisos necesarios para su correcto funcionamiento.

Por ejemplo la aplicación ha de tener acceso a la cámara para el escaneo de códigos o para poder tirar fotos, acceso a internet para poder enviar y recibir los datos, así como para descargarse las actualizaciones cuando estén disponibles o acceso al GPS para el geo-posicionamiento.

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-permission android:name="android.permission.CAMERA" />
<uses-permission android:name="android.permission.VIBRATE" />
<uses-permission android:name="android.permission.FLASHLIGHT" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Ejemplo 4. Permisos de la aplicación

4.4.2 Interfaz de usuario

La interfaz de usuario es el medio que permite al usuario interactuar con la aplicación, es por ello que es importante que dicha interfaz sea fácil e intuitiva de manejar. Esa ha sido la finalidad que se ha intentado buscar en el diseño gráfico de esta aplicación.

La interfaz gráfica de esta aplicación está definida por los ficheros *layouts* que definen el aspecto visual de cada *Activity*.

A la hora de implementar un *layout*, se tienen diferentes conjuntos de vistas con los que trabajar. Un conjunto de vistas distribuye de una manera o de otra los objetos que se van colocando en la pantalla. En el desarrollo de este proyecto se han utilizado algunos de los siguientes conjuntos que aquí se definen:

- **Frame Layout**: el más sencillo de todos, pensado para colocar solamente un objeto en un espacio en blanco (una foto, por ejemplo).
- **Linear Layout**: ubica los objetos unos al lado del otro según la dirección. Solo habrá uno por fila o columna. Seguramente el más apropiado para crear formularios.
- **Table Layout:** éste posiciona los elementos dentro de filas y de columnas formando como su nombre indica una tabla.
- **Absolute Layout:** con él podemos colocar los objetos en cualquier posición de la pantalla, estos utilizan las coordenadas “x” y “y”. En el podemos encontrar que los objetos se solapen unos con otros. Será el utilizado en la aplicación.
- **Relative Layout:** para establecer posiciones relativas entre objetos.

A continuación se va a detallar algunos ejemplos representativos, en los que se puede comprender como están relacionados los *layouts* con las *activities.*

4.4.2.1 Pantalla de acceso

La pantalla de acceso, es la primera pantalla de la aplicación en la cual los usuarios deberán autenticarse. En primer lugar se va mostrar un ejemplo de fichero *layout* completo para poder comprender su estructura. Este *layout* implementa una pantalla sencilla compuesta por el formulario de usuario y contraseña, el icono del proyecto y los botones de acceder y salir, en los sucesivos ejemplos solo se mostrarán los fragmentos de código más representativos. Puede consultarse el aspecto visual en la **ilustración 28** de la *Autenticación de usuario* dentro del manual de usuario posteriormente descrito.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
```

```
    android:layout_height="fill_parent"
    android:background="#FFFFFF"
    android:orientation="vertical" >
    <LinearLayout android:id="@+id/body"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:background="#FFFFFF"
        android:orientation="vertical" >
        <LinearLayout android:id="@+id/linearuser"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="50dip"
            android:background="#FFFFFF"
            android:orientation="horizontal" >
            <TextView android:id="@+id/agente"
                android:layout_width="120dip"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dip"
                android:layout_marginTop="10dip"
                android:text="@string/agente"
                android:textSize="20sp"
                android:textStyle="bold"/>
            <EditText android:id="@+id/codagente"
                android:layout_width="130dip"
                android:layout_height="wrap_content"
                android:layout_marginTop="10dip"/>
        </LinearLayout>
        <LinearLayout android:id="@+id/linearpass"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:orientation="horizontal" >
            <TextView android:layout_width="120dip"
                android:layout_height="wrap_content"
                android:layout_marginLeft="20dip"
                android:text="@string/contrasena"
                android:textSize="20sp"
                android:textStyle="bold" />
            <EditText android:id="@+id/pass"
                android:layout_width="130dip"
                android:layout_height="wrap_content"
                android:password="true"
                android:text="@string/pass"
                android:width="100dip" />
        </LinearLayout>
        <ImageView android:id="@+id/icono"
            android:layout_width="270dp"
            android:layout_height="188dp"
            android:layout_gravity="center_horizontal"
            android:layout_marginTop="20dip"
            android:src="@drawable/iconopfc" />
    </LinearLayout>
    <LinearLayout android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_marginBottom="10dip"
        android:layout_marginTop="10dip"
        android:layout_weight="0"
        android:background="#FFFFFF"
        android:orientation="horizontal" >
        <ImageView android:id="@+id/btnaceptar"
            android:layout_width="40sp"
            android:layout_height="40sp"
            android:layout_marginLeft="50sp"
            android:layout_marginRight="50sp"
            android:src="@drawable/entrar"
            android:text="@string/ok"
            android:textStyle="bold" />
        <ImageView android:id="@+id/btnsalir"
            android:layout_width="40sp"
            android:layout_height="40sp"
            android:layout_marginLeft="75sp"
            android:src="@drawable/salir"
            android:textSize="17sp"
            android:textStyle="bold"/>
    </LinearLayout>
    <TextView android:id="@+id/verIni"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="right"
        android:layout_marginTop="10dip"
        android:text="@string/version"
        android:textSize="12sp" />
</LinearLayout>
```

Ejemplo 5. Codificación del layout acceso.xml

Para mostrar el *layout* arriba definido, deberá relacionarse con su *Activity* correspondiente. El lugar adecuado para realizar esta asociación es el método *onCreate()* de dicha *Activity*.

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.access);

   //edittext del usuario
   txtUsuario = (EditText)findViewById(R.id.codagente);
   //edittext de la contraseña
   txtPass = (EditText)findViewById(R.id.pass);

   //boton imagen aceptar
   ImageView imagenAceptar = (ImageView)findViewById(R.id.btnaceptar);
   //evento click del botón aceptar
   imagenAceptar.setOnClickListener(new View.OnClickListener()
```

```
        public void onClick(View view) {
          aceptar();
        }
    });
    //botón imagen salir
    ImageView imagenSalir = ImageView)findViewById(R.id.btnsalir);
    imagenSalir.setOnClickListener(new View.OnClickListener() {
        public void onClick(View view) {
          salir();
        }
    });
    (…)
}
```

Ejemplo 6. Codificación del método *onCreate()*

La asociación del *layout* con la *Activity* se realiza a través del método *setContentView(),* y de los objetos definidos de dentro de la *Activity* con los definidos en el *layout* se realizan a través del método *findViewById()*. Se puede ver como a los botones de aceptar y salir además de asociarlos a su elemento dentro del *layout* se les asocia un evento a través del método *setOnClickListener()* de forma que cuando se haga *click* sobre alguno de ellos se ejecutará el código asociado al método *onClick()*.

4.4.2.2 Pantalla de listado de clientes

Una vez los usuarios se autentican correctamente se accede a la pantalla principal de la aplicación, esta pantalla muestra un listado con los clientes asociados al usuario. La búsqueda marcada por defecto mostrará los clientes cuya visita está asociada al actual día de la semana, permitiendo también filtrar por rutas, nombre o código de los clientes. Puede consultarse el aspecto visual en la **ilustración 33** en el punto de *Clientes* del manual de usuario.

```
//se rellena el cursor con los clientes
Cursor cursorAux = clienteBuss.FiltrarListaClientes(codigoRuta,
nombrecliente.getText().toString() );
//se indica que el sistema gestione el ciclo de vida del cursor
startManagingCursor(cursorAux);

(…)
// en from[] y to[] se asocian los campos que se van a mostrar
String[] from = clienteBuss.getComlunasListaClientes();
int[] to=new int[]{R.id.ACTIVITY,R.id.CLIENTE,R.id.Visitado,R.id.NoVisitado };

//se crea el elemento grafico de la lista
final ListView list = (ListView) findViewById(R.id.ListViewClientes);
//se registra el menu emergente a la lista de clientes
registerForContextMenu(list);
//se muestra la información a través de PresellerCursorAdapterClientes
```

```
adapter = new PresellerCursorAdapterClientes(this,
R.layout.presellerclientlist, cursorAux, from, to);
list.setAdapter(adapter);
```

Ejemplo 7. Mapeo de datos a través de un *adapter*

Para mostrar el listado de clientes se hace uso de un componente visual de tipo *ListView* definido en el *layout* xml. Para rellenar este tipo de listados se hace uso de las clases denominadas *Adapters* que se encargan de mapear datos en elementos visuales. En este caso los datos de los clientes procedentes de la Base de Datos se almacenan en un objeto de tipo *Cursor* y este es utilizado por el *adapter*, en este caso el *PresellerCursorAdapterClientes*. Este tipo de listados y *adapters* personalizados para mostrar los datos se utilizan a lo largo de toda la aplicación.

Los dos *arrays* que se rellenan y se pasan como parámetro al inicializar el *adapter* hacen referencia a los datos que se quieren mapear tal y como se ha comentado. En este caso el *array from[]* se rellenará con los datos que se quieren mostrar al usuario (los nombres de los campos a los que apunta el *cursor*) y el *array to[]*se rellenará con los elementos visuales que los mostrarán (en este caso los elementos visuales asociados a cada línea de la lista definidos en *layout presellerclientlist*).

Menú emergente de la lista de clientes

Una vez creada la lista de clientes el usuario podrá interactuar con ella, y acceder a los datos de cada una de las líneas que la componen. Para ello se ha definido el evento que permite mostrar el menú emergente con las diferentes acciones que se pueden realizar sobre cada cliente. Puede consultarse el aspecto visual del menú emergente en la **ilustración 35** en el punto de *Selección de clientes* del manual de usuario.

```
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenuInfo
menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   AdapterView.AdapterContextMenuInfo info;
   try {
       info = (AdapterView.AdapterContextMenuInfo) menuInfo;
   } catch (ClassCastException e) {
      return;
   }
   ident = info.position;
   adapter.setSelectedPosition(info.position);
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu_clientes, menu);
}
```

Ejemplo 8. Creación del menú asociado a la lista de clientes

Para capturar los eventos que se producen cuando se pulsa sobre los elementos de la lista, se ha implementado el método *onContextItemSelected()* que permite realizar diferentes acciones dependiendo de la opción del menú emergente que se seleccione. A

continuación se muestra un ejemplo de como se gestionarían dos de las posibles opciones del menú.

```
public boolean onContextItemSelected(MenuItem item) {
   switch (item.getItemId()) {
       //ficha de cliente
       case R.id.item01:
          if(cursorAux.moveToPosition((int) ident)){
              Globales.ClienteCode = cursorAux.getString(1);
              Globales.ClienteName = cursorAux.getString(2);
          }
          Intent intent = new Intent(this, FichaClienteActivity.class);
          Globales.CrearCliente=false;
          this.startActivity(intent);
          return true;
          //pedidos
          case R.id.item02:
              if(cursorAux.moveToPosition((int) ident)) {
                 Globales.ClienteCode = cursorAux.getString(1);
                 Globales.ClienteName = cursorAux.getString(2);
                 Globales.ClienteCodeRoute = cursorAux.getString(6);
                 Globales.ClienteFormaPago = cursorAux.getString(7);
              }
          Intent intenta = new Intent(this, PedidosActivity.class);
          this.startActivityForResult(intenta, 10);
          return true;
(…)
}
```

Ejemplo 9. Gestión de los eventos del menú

Menú principal

Al tratarse de la pantalla principal de la aplicación, desde la pantalla de listado de clientes se mostrará el menú principal de la aplicación. El menú se mostrará cuando se pulse sobre la tecla de menú que ofrecen los dispositivos *Android*. Véase la **ilustración 32** en el punto *Menú Principal* dentro del manual de usuario.

El menú principal se carga implementando el método *onCreateOptionMenu()*.

```
public boolean onCreateOptionsMenu(Menu menu){
       MenuInflater inflater=getMenuInflater();
       inflater.inflate(R.menu.principal_options_menu, menu);
       return true;
}
```

Ejemplo 10. Creación del menú principal de la aplicación

En este caso el menú se ha definido mediante ficheros xml, dentro de la carpeta *res/menu*. Dentro del mismo fichero se pueden crear menús y submenús anidando los elementos correspondientes.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/m_productos"
        android:icon="@drawable/productos"
        android:title="Productos"
        android:visible="true" />
    <item android:id="@+id/m_informe"
        android:icon="@drawable/informs"
        android:title="Informe">
        <menu>
            <item android:id="@+id/m_informeCliente"
                android:title="Informe Cliente" />
            <item android:id="@+id/m_informeProductividad"
                android:title="Informe Productividad" />
        </menu>
    </item>
    (…)
```

Ejemplo 11. Definición del menú principal

Para tratar la opción seleccionada del menú se ha implementado el método *onOptionsItemSelected()*. Dependiendo de la opción que seleccione una vez se muestre el menú, este método se encargará de realizar la acción correspondiente.

```
public boolean onOptionsItemSelected(MenuItem item){
    switch(item.getItemId()){
       //cerrar la aplicacion
       case R.id.m_salir:
             telynetBuss.grabarBaseDatos();
             finish();
        break;
        //productos
       case R.id.m_productos:
             Intent intent = new Intent(this, ProductosActivity.class);
             this.startActivityForResult(intent, 10);
        break;
(…)
```

Ejemplo 12. Captura de los eventos sobre el menú principal

4.4.3 Creación de pedidos

La administración de pedidos es una parte importante del negocio principal de la aplicación, desde la pantalla principal de la aplicación *ClientesActivity* se puede acceder

al módulo de pedidos asociado a cada cliente. En la primera pantalla se accederá a un listado con los pedidos realizados sobre este cliente y que aun no han sido enviados. Véase la **ilustración 41** en el punto *Insertar un nuevo pedido* dentro del manual de usuario.

En cuanto a código Android este módulo no es uno de los mas complejos, pero en cuanto a código JAVA el negocio implementado para la administración de pedidos si ha requerido cierta complejidad, debido a los diferentes factores que han de tenerse en cuenta a la hora de realizar los cálculos, como pueden ser diferentes impuestos aplicables dependiendo de ciertos factores, descuentos de porcentajes o de totales, promociones que añaden productos de regalo al pedido o incrementos en el precio por costes de entrega de ciertos productos.

Se podrán dar de alta pedidos nuevos, editar los pedidos aun no confirmados o borrar los pedidos tanto confirmados como no.

La pantalla de detalle del pedido *NuevoPedidoActivity* muestra un listado con los productos disponibles, que serán los productos que se puedan ir añadiendo al pedido indicando el número de cajas que se quieren añadir. Véase la **ilustración 42** en el punto *Añadir productos al pedido* dentro del manual de usuario.

Desde esta pantalla se podrá tener acceso a la visualización de las promociones y descuentos que el cliente seleccionado tiene disponibles. Estas promociones y descuentos estarán asociados a los diferentes productos de forma que según se vayan añadiendo los productos al pedido se irán calculando. Véase la **ilustración 51** y la **ilustración 52** en el punto *Promociones* y *Descuentos* dentro del manual de usuario.

Cuando se añade un producto al pedido, se comprueba si ese producto tiene alguna promoción asociada, en caso de que solo tenga una se aplica directamente, pero si hay varias promociones se muestra un diálogo emergente para que el usuario decida cual quiere aplicar. Para inicializar el adapter que rellena la lista con las posibles promociones se ha implementado el método *inicializarPromos()*.Véase la **ilustración 46** en el punto *Aplicación de promociones* dentro del manual de usuario.

```
private void inicializarPromos() {
   ListView listapromos =
(ListView)dialogScheme.findViewById(R.id.ListViewRegalos);
   registerForContextMenu(listapromos);
   Context context = getApplicationContext();
   dataPromoAdapter=productoBuss.rellenaAdapterListaSchemes(codProductoSelec,
context);
   listapromos.setAdapter(dataPromoAdapter);
   //evento de item de la lista seleccionado
   listapromos.setOnItemClickListener(new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View view, int position,
long id) {
```

```
            dataPromoAdapter.setSelectedPosition(position);
            aplicarScheme(position, null);
        }
    });
}
```

Ejemplo 13. Inicialización del *adapter* que rellena la lista de promociones

Una vez se han inicializado los datos los datos se muestra el diálogo llamando al método *show()* asociado al objeto del diálogo. Antes de poder mostrar el diálogo se ha debido de implementar el método *onCreateDialog()* en el cual se inicializan todos los objetos *Dialog* que pueda contener la *Activity*.

```
protected Dialog onCreateDialog(int id) {
       switch(id) {
           case DIALOG_CANTIDAD:
               // do the work to define the pause Dialog
              dialogCantidad = new Dialog(this);
              dialogCantidad.setContentView(R.layout.textviewproductos);
              dialogCantidad.getWindow().setBackgroundDrawableResource
(R.color.btncolorgranate);
              dialogCantidad.getWindow().setTitle(descProductoSelec);
              dialogCantidad.getWindow().setTitleColor(Color.rgb(255,255,255));
              inicializarElementos(dialogCantidad);
           return dialogCantidad;

           case DIALOG_REGALOS:
              dialogRegalo1 = new Dialog(this);
              dialogRegalo1.setContentView(R.layout.viewlistaregalos);
              dialogRegalo1.setCancelable(true);
              dialogRegalo1.getWindow().setTitle(descProductoSelec);
              dialogRegalo1.getWindow().setBackgroundDrawableResource
(R.color.btncolorgranate);
              dialogRegalo1.getWindow().setTitleColor(Color.rgb(255, 255,
255));
              inicializarRegalos();
           return dialogRegalo1;
(…)
       }
```

Ejemplo 14. Inicialización del objeto *Dialog* de las promociones

Una vez seleccionada la promoción se añadirán al pedido el productos de regalo asociados, al igual que con las promociones si solo hay un producto asociado se aplicará directamente mientras que si hubiese varios productos se ofrecería un dialogo con las diferentes opciones para que el usuario decidiese. La implementación seguiría la misma dinámica que la utilizada en las promociones.

4.4.4 Geo-localización

La aplicación utiliza la señal GPS para conocer la ubicación real de los clientes, para poder tener controlados los equipos de frío prestados. Se capturan las coordenadas, pudiendo así almacenarlas en la base de datos para posteriormente poder enviarlas al servidor.

Para poder acceder y controlar a la señal GPS se utilizan las clases presentes en el paquete *android.location*. En general, este paquete incluye clases que permiten manejar diferentes dispositivos de localización. Algunos de los elementos presentes en este paquete son:

- *Location*: clase que representa una localización geográfica.
- *LocationManager*: clase para gestionar el dispositivo de localización.
- *LocationListener*: interfaz que permite implementar una clase que captura los eventos asociados al dispositivo de localización.

Las clases básicas para obtener la señal de GPS son *LocationManager* para controlar el dispositivo, *LocationListener* para escuchar sus cambios y *Location* para guardar la información de localización. La clase *EquiposFrioActivity* implementa el interfaz *LocationListener* está declarada una variable global *locationManager* para controlar el dispositivo, e implementar los eventos asociados al GPS.

En primer lugar, es necesario acceder al dispositivo de localización, el GPS en este caso. Para acceder a este o a cualquier otro servicio integrado en el dispositivo móvil, se utiliza el método *getSystemService()* la clase *Activity* y sus clases derivadas, en este caso es *EquiposFrioActivity*. Para referenciar al GPS es necesario pasar la constante Context.LOCATION_SERVICE. De esta forma, en *locationManager* se tendrá un controlador válido para el dispositivo GPS.

```
LocationManager locationManager = (LocationManager) getSystemService
(Context.LOCATION_SERVICE);
```

Ejemplo 15. Inicialización del objeto *LocationManager*

Al implementar el interfaz *LocationListener* es necesario implementar los métodos *onLocationChanged()*, *onProviderDisabled()*, *onProviderEnabled()* y *onStatusChanged()* desde los que se capturan los eventos relacionados con el GPS.

Además, es necesario otorgar a la aplicación permiso correspondiente para acceder al dispositivo de localización mediante la declaración en el fichero AndroidManifest.xml de las etiquetas correspondientes:

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

Ejemplo 16. Permisos necesarios para la utilización del GPS

A continuación se puede ver el código, desde el método *lanzarCapturaGPS()* se inicia la captura de coordenadas, una vez se capturen se lanzará el evento *onLocationChanged()* en el que se recogen y se llama al método *mostrarMensajeCoordenadas()* que se encarga de preguntar al usuario si quiere almacenar la coordenadas capturadas en la base de datos.

```
@Override
public void onLocationChanged(Location location) {
   latitud = location.getLatitude();
   longitud = location.getLongitude();
   mostrarMensajeCoordenadas(longitud, latitud);
}
private void lanzarCapturaGPS(){
   Criteria criteria = new Criteria();
   provider = locationManager.getBestProvider(criteria, true);
   if(provider != null){
       Toast.makeText(getApplicationContext(),
getString(R.string.msgCapturandoCoordenadas), Toast.LENGTH_SHORT).show();
       List<String> listaProviders = locationManager.getProviders(true);
       if(listaProviders.size() <= 0)
          Toast.makeText(this, getText(R.string.msgDisabledProviders),
Toast.LENGTH_SHORT).show();
          locationManager = (LocationManager) getSystemService
(Context.LOCATION_SERVICE);
          locationManager.requestLocationUpdates(provider, 0, 0, this);
       }else{
          Toast.makeText(getApplicationContext(),
getString(R.string.msgProveedoresGPSDeshabilitado),Toast.LENGTH_SHORT).show();
       }
 }
```

Ejemplo 17. Captura de coordenadas GPS

4.4.5 Escáner de códigos QR y códigos de barras

Como se ha visto en los requisitos, la aplicación permite al usuario escanear códigos QR directamente desde el dispositivo. Para tal fin, se han utilizado las librerías de *zxing* (Zxing, 2012). Zxing es una librería para el procesado de imágenes de códigos de barra implementada en JAVA y que, además, es de código abierto, por lo que su uso es libre por parte de los usuarios.

A la hora de utilizar dicha librería en cualquier proyecto existen dos posibilidades. Por una parte, se podría utilizar la aplicación propia de *zxing*, llamada *BarcodeScanner* y que está disponible en el *Market* o *Google Play* (Google Play, 2012), o bien utilizar el código fuente disponible e incluir las necesidades propias del proyecto. Para la realización de este proyecto se ha optado por la segunda opción, pues la primera implica

que los dispositivos tengan acceso a otras aplicaciones y, dada la naturaleza empresarial del proyecto, esto puede que no se dé siempre.

A continuación se explicará el proceso necesario para que sea posible el escaneo de códigos, desde la llamada desde el proyecto a esta librería, así como el procesamiento del resultado obtenido.

Para la realización del escáner, se ha creado un nuevo proyecto en eclipse que actuará como librería del proyecto principal. Este proyecto, llamado sdk_*scannercodigos*, es un proyecto Android de tipo librería, por lo que dispondrá también del fichero de *manifest*, así como de sus propias *Activities*. Se ha decidido hacerlo así para facilitar la reutilización de este código, a la hora de añadírselo a alguna otra aplicación que también tenga que leer códigos QR o códigos de barras.

A continuación se muestra el contenido del *manifest* de la aplicación. Se puede apreciar que el proyecto cuenta una *Activity* llamada *CameraCaptureActivity*, que será la principal de la aplicación. Ésta será la que se llame desde el proyecto principal y se encargará de inicializar el resto de elementos necesarios para el escaneo de códigos.

También cabe destacar los permisos necesarios para el correcto funcionamiento de la aplicación. Se necesita acceso a la cámara del teléfono, a la función de vibración, al flash de la cámara y se necesita permiso de escritura en el sistema de almacenamiento externo para almacenar los resultados del escaneo.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="droid.preseller.sdk "
    android:versionCode="1" android:versionName="1.0" >
    <uses-sdk android:minSdkVersion="7" />
    <application android:icon="@drawable/ic_launcher"
       android:label="@string/app_name" >
       <activity android:name="droid.preseller.sdk.CameraCaptureActivity"
             android:label="@string/app_name"
             android:screenOrientation="landscape">
             <intent-filter >
                 <action android:name="android.intent.action.MAIN" />
                 <categoryandroid:name="android.intent.category.LAUNCHER" />
             </intent-filter>
     </activity>
  </application>
  <uses-permission android:name="android.permission.CAMERA" />
  <uses-permission android:name="android.permission.VIBRATE" />
  <uses-permission android:name="android.permission.FLASHLIGHT" />
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
</manifest>
```

Ejemplo 18. Llamada y permisos para la captura de códigos

CameraCaptureActivity

Es la *Activity* que se llama desde el proyecto principal. Se encarga de inicializar todos lo elementos necesarios para el escaneo del códigos. Una vez que se ha escaneado el código, éste se devolverá a la *Activity* origen que la lanzó.

La inicialización de los componentes ocurre, como suele ser habitual, en el método *onCreate()* de la *Activity*.

```
@Override
public void onCreate(Bundle icicle) {
   super.onCreate(icicle);
   Window window = getWindow();
   window.addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
   setContentView(R.layout.camera_capture);
   CameraManager.init(getApplication());
   viewfinderView=(ViewfinderView)findViewById
       (R.id.camera_view_viewfinder_view);
   handler = null;
   hasSurface = false;
   preinitCamera();
}
```

Ejemplo 19.Codificación del método *onCreate()* de *CameraCaptureActivity*

Además de las labores habituales de crear el interfaz gráfico (a partir de un *layout* xml) y obtener referencias a ellos, son aspectos importantes de este método la inicialización del objeto *CameraManager*, y posteriormente la pre-inicialización de la cámara del dispositivo.

El objeto *CameraManager*, envuelve el objeto servicio de la cámara. La implementación encapsula los pasos necesarios para el *preview* de imágenes necesarios para la decodificación. También se encarga de labores como abrir y cerrar el driver apropiado de la cámara y de calcular el rectángulo donde el usuario deberá poner el código tanto de barras como de tipo QR para su procesamiento.

```
private void preinitCamera() {
SurfaceView surfaceView = (SurfaceView)
findViewById(R.id.camera_view_preview_view);
   SurfaceHolder surfaceHolder = surfaceView.getHolder();
   if (hasSurface) {
       initCamera(surfaceHolder);
   } else {
       surfaceHolder.addCallback(this);
       surfaceHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
   }
}
```

Ejemplo 20. Pre inicialización de la cámara

Como se puede observar, se utiliza un objeto de tipo *SurfaceView* para mostrar el contenido procesado por la cámara. El objeto *SurfaceView* proporciona una superficie dedicada donde se puede dibujar dentro de una jerarquía de vistas establecidas. Se puede controlar el formato de la superficie, así como el tamaño. El objeto *SurfaceView* se ocupa de mostrar el contenido en la posición adecuada en la pantalla.

El otro aspecto importante de este método sería la llamada a *initCamera()*.

```
private void initCamera(SurfaceHolder surfaceHolder) {
   try {
       CameraManager.get().openDriver(surfaceHolder);
   } catch (IOException ioe) {
       Log.w(TAG, ioe);
       displayFrameworkBugMessageAndExit();
       return;
   } catch (RuntimeException e) {
       Log.w(TAG, "Unexpected error initializating camera", e);
       displayFrameworkBugMessageAndExit();
       return;
   }
   if (handler == null) {
       handler=new CameraCaptureActivityHandler(this,decodeFormats,
characterSet);
   }
}
```

Ejemplo 21. Inicialización de la cámara

Este método inicializa la cámara abriendo el driver e inicializando los parámetros hardware. Además, crea la instancia de la clase *CameraCaptureActiviyHandler*. Esta clase, de la que se hablará a continuación, se encarga de inicializar el hilo encargado de la decodificación en sí misma, y hará de puente (mediante un sistema de mensajes) entre dicho hilo decodificador y la *Activity CameraCaptureActivity*. Este proceso de mensajes entre el hilo decodificador y el objeto *CameraCaptureActivityHandler* desemboca en la llamada al método *handleDecode()* de *CameraCaptureActivity*.

```
public void handleDecode(Result rawResult, Bitmap barcode) {
  CharSequence dis = getDisplayContents(rawResult);
  String scanCod = dis.toString();
  String pathFichero = "";
  String nombreFichero = "";
  if(GlobalesScannerCodigos.guardarBitmapCodigo){
      nombreFichero = GlobalesScannerCodigos.nombreFicheroCapturado;
      if(nombreFichero != null && !nombreFichero.trim().equals("")){
          pathFichero = GlobalesScannerCodigos.pathImagenes + nombreFichero;
      }else{
          nombreFichero = scanCod+".png";
```

```
            pathFichero = GlobalesScannerCodigos.pathImagenes + nombreFichero;
        }
        try {
            FileOutputStream out = new FileOutputStream(pathFichero);
            barcode.compress(Bitmap.CompressFormat.PNG, 90, out);
        } catch (Exception e) {
            Log.w(TAG, "error al guardar la imagen de codigo ", e);
        }
    }
    Intent resultIntent = new Intent();
    resultIntent.putExtra(GlobalesScannerCodigos.ExtraScanCod, scanCod);
    resultIntent.putExtra(GlobalesScannerCodigos.ExtraScanBitmap,nombreFichero);
    setResult(Activity.RESULT_OK, resultIntent);
    finish();
    setSound(R.raw.beep);
    playSoundAndVibrate();
}
```

Ejemplo 22. Captura del código

Este método hace varias cosas importantes. En primer lugar, se encarga de transformar el resultado del escaneo (objeto *Result*) en un *String*, de manera que la aplicación lo pueda manejar fácilmente. Además, almacena en el sistema de ficheros de Android la imagen capturada si así se le indica a través de una variable *boolean*. Finalmente, se encarga de devolver el resultado del escaneo del código a la *Activity* que originó la llamada.

Como se ve en las líneas de código expuestas arriba, para devolver el resultado no hace falta más que crear un nuevo *Intent* donde se añadirán los parámetros que se devolverán en el resultado. Finalmente, una llamada al método *finish()* indicando que la *Activity* puede darse por finalizada.

CameraCaptureActivityHandler

CameraCaptureActivityHandler es otro de los elementos importantes dentro de todo el proceso de escaneo de códigos. Como se ha mencionando anteriormente, se ocupa de inicializar el hilo encargado del proceso de decodificación de las imágenes capturadas y hace de puente entre este hilo y la *Activity* donde se procesa el resultado. En este apartado se explican los aspectos más importantes de dicha clase. Hay dos elementos clave que merecen ser explicados dentro de la clase. El primero de ellos es el constructor de la misma:

```
CameraCaptureActivityHandler(CameraCaptureActivity
activity,Vector<BarcodeFormat> decodeFormats, String characterSet) {
   this.activity = activity;
   decodeThread = new DecodeThread(activity, decodeFormats, characterSet,
             new ViewfinderResultPointCallback(activity.getViewfinderView()));
   decodeThread.start();
   state = State.SUCCESS;
```

```
    // Comienza la captura y la decodificación
    CameraManager.get().startPreview();
    restartPreviewAndDecode();
}
```

Ejemplo 23. Constructor de *CameraCaptureActivityHandler*

Como ya se ha visto, este constructor es llamado desde el método que inicializa la cámara para comenzar con la decodificación. Entre los distintos parámetros que le llegan al constructor destaca la instancia de la Activity *CameraCaptureActivity*, de modo que se tenga acceso a los métodos de ésta. Esta instancia se le pasará además al hilo *DecodeThread*, de modo que la comunicación entre ambos sea posible. Dentro del constructor también cabría destacar la inicialización y ejecución del hilo *DecodeThread*, encargado del procesamiento de la imagen para su decodificación.

El otro aspecto fundamental de la clase es el método *handleMessage()*, que es donde se reciben los mensajes provenientes del hilo decodificador *DecodeThread*. Estos mensajes consistirán en un código (error, éxito, etc.) y unos datos (el código escaneado).

```
public void handleMessage(Message message) {
    if (message.what == R.id.auto_focus) {
        if (state == State.PREVIEW) {
            CameraManager.get().requestAutoFocus(this, R.id.auto_focus);
        }
    } else if (message.what == R.id.restart_preview) {
        Log.d(TAG, "Got restart preview message");
        restartPreviewAndDecode();
    } else if (message.what == R.id.decode_succeeded) {
            Log.d(TAG, "Got decode succeeded message");
            state = State.SUCCESS;
            Bundle bundle = message.getData();
            Bitmap barcode = bundle == null ? null : (Bitmap) bundle
                .getParcelable(DecodeThread.BARCODE_BITMAP);
            activity.handleDecode((Result) message.obj, barcode);
        } else if (message.what == R.id.decode_failed) {
                state = State.PREVIEW;
               CameraManager.get().requestPreviewFrame(decodeThread.getHandler(),
        R.id.decode);
        }
}
```

Ejemplo 24. Codificación del método *handleMessage()*

El procedimiento, por tanto, consistirá en procesar dichos mensajes que llegarán continuamente, comprobar si se ha escaneado algún código y, de ser así, llamar al método de la *Activity* que se encarga de procesar y devolver el resultado. Si, por el contrario, no se ha escaneado ningún código, se descartará el *preview* y se empezará el proceso de nuevo.

La llamada desde el proyecto principal a las funcionalidades de esta librería, concretamente a la *Activity CameraCaptureActivity*, no difiere mucho de una llamada a

una *Activity* cualquiera. Al igual que en el resto de llamadas, habrá que crear un objeto de tipo *Intent* con la información necesaria para iniciar la *Activity*. Además, dado que es necesario obtener como respuesta el resultado del código escaneado, habrá que implementar el método *onActivityRestult()* en la *Activity* origen. A continuación se muestran las partes del código más relevantes de este proceso.

```
GlobalesScannerCodigos.nombreFicheroCapturado = nombreCodigo;
GlobalesScannerCodigos.guardarBitmapCodigo = false;
Intent intentc = new Intent(this, CameraCaptureActivity.class);
startActivityForResult(intentc, R.id.escaner_codigos);
```

Ejemplo 25. Llamada a la *Activity* que captura los códigos

Como se puede apreciar, la llamada es aparentemente igual a la realizada para iniciar cualquier otra *Activity*. Para obtener el resultado del escaneo de código se utilizará el método *onActivityResult()*. Este método es llamado cuando la *Activity* que se lanzó ha finalizado su proceso, siendo el lugar adecuado cuando para obtener y procesar el posible resultado.

```
protected void onActivityResult(int requestCode, int resultCode, Intent data){
super.onActivityResult(requestCode, resultCode, data);
switch(requestCode) {
  case R.id.escaner_codigos :
    if (resultCode == Activity.RESULT_OK) {
       String codScan =
           data.getStringExtra(GlobalesScannerCodigos.ExtraScanCod);
       String nombreFotoBitmapScan =
           data.getStringExtra(GlobalesScannerCodigos.ExtraScanBitmap);
       codigoCapturado.setText(codScan);
       codigoCapturado.setVisibility(View.VISIBLE);
       if(GlobalesScannerCodigos.guardarBitmapCodigo){
          if(nombreFotoBitmapScan!=null &&
          !nombreFotoBitmapScan.trim().equals("")){
              equiposFrioBus.borrarCodigoAntiguo(codEquipoFrio);
              equiposFrioBus.insertarFoto(nombreFotoBitmapScan, codEquipoFrio,
              false);
          }
        }
     }
break;
 (…)
}
```

Ejemplo 26. Captura del resultado capturado

Entre los parámetros que recibe este método, aparte de los datos de la respuesta, es muy importante el parámetro *requestCode*. Este parámetro es el mismo que se utilizó para iniciar la llamada a la *Activity* y es fundamental para poder distinguir qué resultado es el que estamos obteniendo. Ha de tenerse en cuenta que todos los resultados de las

distintas *Activities* que se lancen desde ésta desembocarán en este método, por lo que es muy importante poder distinguir qué resultado se ha de procesar.

4.4.6 Preferencias

Las preferencias de la aplicación no son más que datos de configuración que se utilizan para personalizar la experiencia del usuario, como pueden ser la dirección del servidor, la ruta donde se quieren guardar los ficheros de logs de la aplicación o el tiempo de espera que se quiere permitir cuando se realiza una comunicación. Véase la **ilustración 77 y la ilustración 78** de la *Configuración* dentro del manual de usuario.

Las preferencias de una aplicación se podrían almacenar en Base de Datos, pero Android proporciona este método alternativo diseñado específicamente para administrar este tipo de datos, las denominadas preferencias compartidas o *shared preferences*, del paquete *android.preference.PreferenceActivity*.

Cada preferencia se almacenará en forma de clave-valor, es decir, cada una de ellas estará compuesta por un identificador único y un valor asociado a dicho identificador. Además, y a diferencia de SQLite, los datos no se guardan en un fichero de binario de base de datos, sino en ficheros XML.

En este proyecto se han creado tres ficheros de preferencias, para poder estructurarlas en preferencias generales, configuración del servidor y configuración de ficheros, ya que debido al número de datos que se querían configurar, en una sola pantalla quedaría demasiado extensa. Las clases que las implementan son muy sencillas, ya que únicamente agregan las propiedades especificadas en los *layouts*.

```
public class PresellerPreferencesActivity extends PreferenceActivity {
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		addPreferencesFromResource(R.xml.preferences);
	}
}
```

Ejemplo 27. Carga de los valores por defecto de las preferencias

Cuando se inicia la aplicación se quiere que se carguen los valores por defecto definidos en las preferencias. Para ello se deben cargar desde el método *onCreate()* de la *Activity* inicial.

```
//se inicializana los valores por defecto de las preferences
PreferenceManager.setDefaultValues(this, R.xml.preferences, true);
PreferenceManager.setDefaultValues(this, R.xml.preferences2, true);
PreferenceManager.setDefaultValues(this, R.xml.preferences3, true);
```

Ejemplo 28. Inicialización de las preferencias

Para leer los valores almacenados en las preferencias, se deberán recuperar desde el código de la aplicación a través del objeto *SharedPreferences*.

```
SharedPreferences preferences=
PreferenceManager.getDefaultSharedPreferences(context);
(…)
GlobalesPreseller.DIR_APP_BACKUP = preferences.getString("pathAppBackup","");
```

Ejemplo 29. Recuperación de uno de los valores de las preferencias

4.4.7 Construcción y acceso a base de datos

Como ya se ha comentado anteriormente los datos de la aplicación se almacenan en una Base de datos SQLite. La gestión de acceso a la Base de Datos se lleva a cabo en varias clases, por una parte se tendría la clase de inicialización y por otra las clases que nos facilitan el acceso a los datos desde la aplicación.

La base de datos inicial de la aplicación se ha creado previamente con la aplicación SQLite Manager. Una vez la Base de Datos está creada se debe añadir al proyecto en la carpeta *assets* que será donde la aplicación vaya a buscarla. Cuando se inicia la aplicación el código se encarga de comprobar si el fichero *.sqlite* existe y si no existe realizaría una copia de la Base de Datos inicial en el sistema de ficheros de Android. De esta forma la base de datos solo se copiará la primera vez que se ejecute la aplicación. Así se asegura que aunque se instalen actualizaciones de la aplicación la base de datos nunca se va a sobrescribir.

La clase que se encarga del procedimiento inicial de la Base de Datos *DataBaseGeneral* que extiende de la clase *SQLiteOpenHelper*, a continuación se puede ver como es el proceso en el que se copia el fichero con la Base de Datos a la ruta apropiada de la aplicación.

```
public void crearDataBase() throws IOException {
	boolean dbExist = checkDataBase();
	if (!dbExist) {
		this.getReadableDatabase();
		try {
			copiarBaseDatos();
		} catch (IOException e) {
			throw new Error("Error copiando base de datos");
		}
	}
}
(…)
private void copiarBaseDatos() throws IOException{
```

```
        InputStream myInput =
myContext.getAssets().open(GlobalesDefinicionDatos.DB_NAME);
        String outFileName = GlobalesDefinicionDatos.DB_PATH +
GlobalesDefinicionDatos.DB_NAME;
        OutputStream myOutput = new FileOutputStream(outFileName);
        byte[] buffer = new byte[1024];
        int length;
        while ((length = myInput.read(buffer))>0){
              myOutput.write(buffer, 0, length);
        }
        myOutput.flush();
        myOutput.close();
        myInput.close();
}
```

Ejemplo 30. Copia de la Base de Datos

Toda la lógica de acceso a datos de la aplicación se ha implementado en la clase *PresellerAccesoDatosGeneralDB* que extiende de la clase *SQLiteOpenHelper*. Esta clase tiene un objeto de tipo *SQLiteDatabase* necesario para la consulta, inserción y borrado de registros en la Base de Datos. Todos los métodos de acceso a la Base de Datos se han implementado en esta clase, de forma que cada una de las clases asociadas a cada una de las tablas extiendan de ella y así puedan tener acceso a todos los métodos sin necesidad de tener que volverlos a implementar. Todas las clases asociadas a las tablas estarán dentro del paquete *com.droid.preseller.dataaccess*.

```
public class PresellerAccesoDatosGeneralDB extends SQLiteOpenHelper{
        private SQLiteDatabase myDataBase;
        private String DATABASE_TABLE;
(…)
}
```

Ejemplo 31. Declaración de la clase *PresellerAccesoDatosGeneral*

La clase *PresellerAccesoDatosGeneralDB* implementa todos los métodos necesarios para generar consultas, ejecutar las consultar, hacer inserciones y borrado de registros. Un ejemplo de una clase asociada a una tabla sería la clase *PedidosCabeceraDB* asociada a la tabla *orders_headers*, donde se almacenan la cabecera de los pedidos.

```
public class PedidosCabeceraDB extends PresellerAccesoDatosGeneralDB{
        /** Constante con el nombre de la tabla*/
        private static final String DATABASE_TABLE = "orders_headers";
        /** Array en el que se almacenan los nombres de las columnas*/
        public static final String [] columnas  = new String[] {
        "status_mobile","cod_emp","code_route","cod_cliente", "numerodoc",
        (…)    };
    /** Variables asociadas a los nombres de las columnas de la tabla*/
```

```
        public final String KEY_STATUSMOBILE = columnas[0];
        public final String KEY_COD_EMP = columnas[1];
        public final String KEY_CODE_ROUTE = columnas[2];
        public final String KEY_COD_CLIENTE = columnas[3];
        public final String KEY_NUMERO_DOC = columnas[4];
        (…)
}
```

Ejemplo 32. Codificación de una clase asociada a una tabla

Para insertar registros en al Base de Datos, se ha de obtener una instancia del objeto *SQLiteDatabase*, en este caso al estar extendiendo de la clase *PresellerAccesoDatosGeneralDB* ya se tendría asociado a todas las tablas. A través del método *myDataBase.insert()*se realizará la inserción en la columna correspondiente. A este método es necesario pasarle un objeto *ContentValue* que contendrás los valores y las columnas que se quieren rellenar. En el siguiente ejemplo se puede ver el método *insertarRegistro()* implementado en la clase *PresellerAccesoDatosGeneralDB* al que se le pasan dos *arrays* con los nombres de las columnas y los valores que se quieren insertar y se encarga de generar el objeto *ContentValue* correspondiente. Este método devolverá un entero con el número de la nueva fila insertada o -1 en cas de error.

```
public long insertarNuevoRegistro(String keys[], String values[]) throws
SQLiteException{
   ContentValues initialValues = new ContentValues();
   if (keys.length == values.length && keys.length >0){
       for(int i = 0 ; i<keys.length ; i++){
          String valor = values[i];
           if(valor != null )
              initialValues.put(keys[i],values[i]);
       }
   }
   long resultadoInsercion = 0;
   try{
      resultadoInsercion=myDataBase.insert(DATABASE_TABLE,null,initialValues);
   }catch(SQLiteException e){
      throw e;
   }
   return resultadoInsercion;
}
```

Ejemplo 33. Inserción de un registro en la Base de Datos

Para borrar registros, se procede de una forma muy similar a la inserción. A través del objeto *SQLiteDatabase* se puede realizar la llamada al método *myDataBase.delete()* que se encarga de borrar los registros correspondientes. Para este caso se ha implementado el método *borrarRegistro()* que devuelve *true* si el registro se ha borrado con éxito y *false* en caso contario.

```
public boolean borrarRegistro(String whereClause, String whereArgs[])throws
SQLiteException{
   boolean resultadoBorrado = false;
   try{
       long borrados= myDataBase.delete(DATABASE_TABLE whereClause, whereArgs);
       resultadoBorrado = borrados > 0;
   }catch(SQLiteException e){
        throw e;
    }
       return resultadoBorrado;
}
```

Ejemplo 34. Borrado de un registro en la Base de Datos

Para realizar consultar sobre la Base de Datos se han implementado varios métodos, debido a la necesidad de la realización de los diferentes tipos de *querys*, entre ellos se han utilizado las llamadas a los métodos *myDataBase.execSQL(), myDataBase.rawQuery() o sQLiteQueryBuilder.query().* En el fragmento de código posterior se muestra un ejemplo de búsqueda sobre la Base de Datos, en estos métodos se va a devolver un objeto de tipo *Cursor* para que posteriormente le negocio de la aplicación lo trate en cada caso correspondiente.

```
public Cursor ejecutaQuery(boolean distinct, String tabla, String[] columns,
String selection, String[] selectionArgs,String where, String groupBy, String
having, String orderBy,String limit)throws SQLiteException {

   Cursor cursor = null;
   try{
       String sql = SQLiteQueryBuilder.buildQueryString(distinct, tabla,
columns, where, groupBy, having, orderBy, limit);
       cursor = myDataBase.rawQuery(sql, selectionArgs);
   }catch(SQLiteException e){
       throw e;
   }
   return cursor;
}
```

Ejemplo 35. Ejecución de una query sobre la Base de Datos

4.4.8 Comunicaciones

Las comunicaciones forman una parte importante de la aplicación, ya que van a permitir el envío y la recepción de datos para el funcionamiento vivo de la aplicación. Desde el menú principal de la aplicación se puede acceder a la opción de comunicaciones, esta opción se va a encargar de realizar el proceso completo que implicaría una comunicación.

Como se comentó en el diseño de la aplicación todas las tablas tienen una columna denominada *status_mobile* que va a marcar el estado de ese registro a la hora de decidir si se va a enviar.

El proceso completo de la comunicación estaría divido en las siguientes acciones:

- Escritura de los registros modificados o insertados en la Base de Datos al fichero de exportación.
- Subida del fichero de exportación y del fichero de fotos si fuera necesario al buzón de usuario habilitado para este fin en el servidor.
- Chequeo del buzón de bajada del usuario para comprobar si tiene algún fichero con datos que descargar.
- En caso de que haya fichero para descargar, descarga del mismo.
- Si se ha producido descarga de fichero se importará en la Base de Datos.
- Una vez finalizado con éxito el proceso de importación se realizara la actualización de estados en la Base de Datos para indicar que los registros modificados o insertados ya se han enviado y no volverlos a enviar en la siguiente comunicación.

La lógica de este proceso está implementada en la clase *PresellerComunicacionesBusinessRules* que es la que se va a encargar de ir haciendo las diferentes llamadas para llevar a cabo el ciclo de comunicación anteriormente explicado. Para realizar este proceso esta clase extiende PresellerAsyncTask, de esta forma al tratarse de una *AsyncTask* el negocio estará implementado en el método *doInBackground()* que va a permitir ir actualizando los mensajes que se van mostrando al usuario para indicarle como esta transcurriendo la comunicación.

La primera tarea que se va a realizar en el proceso de comunicación es la exportación de los registros modificados, borrados o insertados nuevos a un fichero de texto para su posterior envío. En primer lugar se crea el fichero y se le introducen la cabecera correspondiente. De la tabla *TDTablas* se obtienen los nombres de las tablas de exportación, y de las tablas *TMAEDiccionarioDatos* se obtienen los nombres de los campos de estas tablas. Finalmente se va recorriendo cada una de estas tablas para buscar los registros que tengan el campo *status_mobile* marcado para exportar y según se vayan encontrando se van añadiendo al fichero.

Una vez se ha generado el fichero de exportación se procede a subirlo al servidor, en caso de que no hubiera registros que exportar se subiría un fichero de exportación compuesto únicamente por la cabecera. La subida del fichero se hace a través de una llamada *HTTP* de tipo *POST* tal y como se muestra a continuación.

```
public int downloadEnviarFichero(String accion,String buzon,String
nombreFichero){
   int resultado = -1;
   String parametros =
GlobalesComunicaciones.comienzoParametro+GlobalesComunicaciones.parametroAccio
```

```
n+accion+GlobalesComunicaciones.separacionParametros+GlobalesComunicaciones.pa
rametroPalmtop+buzon+GlobalesComunicaciones.separacionParametros+GlobalesComun
icaciones.parametroFile+nombreFichero;

   HttpURLConnection connection = null;
   DataOutputStream outputStream = null;
   int bytesRead, bytesAvailable, bufferSize;
   byte[] buffer;
   int maxBufferSize = 1*1024*1024;
   try {
       File file = new File(GlobalesComunicaciones.FILES_PATH +nombreFichero);
       sizeFichLocal = (int) file.length();
       FileInputStream fileInputStream = new FileInputStream(file);

       URL url = new URL(urlServer + urlFileDownload + parametros);
       int timeoutConnection = GlobalesComunicaciones.timeoutConnection;
       int timeoutSocket = GlobalesComunicaciones.timeoutSocket;
       connection = (HttpURLConnection) url.openConnection();
       connection.setReadTimeout(timeoutConnection+50000);
       connection.setConnectTimeout(timeoutSocket+50000);
       connection.setDoInput(true);
       connection.setDoOutput(true);
       connection.setUseCaches(false);
       connection.setRequestMethod("POST");
       connection.setRequestProperty("Connection", "Keep-Alive");
       connection.setRequestProperty("Content-Type", "multipart/form-data");
       outputStream = new DataOutputStream( connection.getOutputStream() );
       bytesAvailable = fileInputStream.available();
       bufferSize = Math.min(bytesAvailable, maxBufferSize);
       buffer = new byte[bufferSize];
       bytesRead = fileInputStream.read(buffer, 0, bufferSize);

       while (bytesRead > 0){
          outputStream.write(buffer, 0, bufferSize);
          bytesAvailable = fileInputStream.available();
          bufferSize = Math.min(bytesAvailable, maxBufferSize);
          bytesRead = fileInputStream.read(buffer, 0, bufferSize);
       }
       resultado = connection.getResponseCode();
       fileInputStream.close();
       outputStream.flush();
       outputStream.close();
   }
   catch (SocketTimeoutException e) { resultado = -1; }
   catch (ConnectTimeoutException e){ resultado = -1; }
   catch (IOException e)  { resultado = -1;   }
   catch (Exception e)    {resultado = -1;    }
   return resultado;
}
```

Ejemplo 36. Envío de un fichero

Una vez subido el fichero de exportación se procede a ver si hay un fichero de importación que descargar. El fichero siempre se va a encontrar en la misma *url* por lo

que el método encargado de realizar la descarga compondrá la *url* completa y se bajará el fichero al dispositivo a través de una conexión *HTTP* utilizando un método *GET*.

```
public int downloadRecibirFichero(String accion, String buzon, String
urlDescargaFichero, String nombrefichero){
   int resultado = -1;
   try {
       int timeoutConnection = GlobalesComunicaciones.timeoutConnection;
       int timeoutSocket = GlobalesComunicaciones.timeoutSocket;
       URL u = new URL(urlDescargaFichero + nombrefichero);
       HttpURLConnection c = (HttpURLConnection) u.openConnection();
       c.setReadTimeout(timeoutConnection+50000);
       c.setConnectTimeout(timeoutSocket+50000);
       c.setRequestMethod("GET");
       c.setDoOutput(true);
       c.connect();
       int lenghtOfFile = c.getContentLength();
      InputStream in = c.getInputStream();
      File file = new File(GlobalesComunicaciones.FILES_PATH, nombrefichero);
      FileOutputStream f = new FileOutputStream(file);
      byte[] buffer = new byte[1024];
      int len1 = 0;
      long total = 0;

      while ((len1 = in.read(buffer)) > 0) {
          total += len1; //total = total + len1
          f.write(buffer, 0, len1);
       }
       f.flush();
       f.close();
       in.close();
       if(total > 0)
          resultado = (int) file.length();
        else
          resultado = lenghtOfFile;
   }
   catch (SocketTimeoutException e){    resultado = -1; }
   catch (ConnectTimeoutException e){   resultado = -1; }
   catch (FileNotFoundException e){    resultado = -1; }
   catch (IOException e){    resultado = -1;  }
   catch (Exception e) {  resultado = -1; }
   return resultado;
}
```

Ejemplo 37. Recepción de un fichero

Una vez descargado el fichero en caso de haberlo se procederá a importarlo siguiendo una lógica parecida a la que se utilizo para realizar el fichero de exportación. Se llama al método *leerFichero()* que devuelve un objeto de tipo *ArrayList* con cada una de las líneas del fichero. Se va tratando línea a línea de forma que cuando se indica

el nombre de una tabla se obtiene de la tabla *TMADiccionarioDatos* los nombres de los campos de la tabla y cuales son la *Primary Keys* o *PK*, para poder componer la *query* de inserción. Una vez que se tienen las PK de la tabla, se compone la *query* y se llama al método *InsertarRegistro()* que intenta hacer una actualización del campo por si ese registro ya estuviese en la Base de Datos y no está realiza una inserción con los métodos comentados en el apartado anterior de acceso a Base de Datos.

Una vez importado el fichero descargado en caso de que lo hubiere, si la comunicación se ha realizado sin errores se procede actualizar los estado de la Base de Datos, para ello se recorre las tablas obtenidas en el método *obtenerNombreTablas()* y todos aquellos registros marcados con un *status_mobile* con estado de envío se actualizarían como que ya se han enviado con éxito para no enviarlos en la siguiente comunicación. Si se ha producido algún error en la comunicación no se ejecutaría la actualización de estados, ya que los registros no se han enviado con éxito.

4.5 Manual de usuario

En este apartado se describe el funcionamiento básico de la aplicación acompañado de capturas reales de la imagen final que tiene la aplicación en el momento de finalización de este proyecto. Este apartado contiene a descripción de la interfaz de usuario y define cada función o proceso que forman la aplicación.

La aplicación se encarga de englobar las necesidades de un técnico pre-venta de una compañía encargado de visitar a sus clientes realizando sus labores comerciales, facilitándole las herramientas necesarias para realizar su trabajo. Con el sistema ofrecido el técnico pre-venta podrá realizar las siguientes funciones.

- Visitar a los clientes de acuerdo a la ruta programada para el día de la semana en que se encuentra, o modificar dichas rutas para añadir o quitar clientes si fuera necesario.
- Consultar y modificar la información general de los clientes.
- Consultar el histórico con los últimos pedidos realizados por cada cliente.
- Tomar nota de los nuevos pedidos realizados por el cliente.
- Aplicación de descuentos y promociones a la hora de realizar el pedido.
- Añadir comentarios asociados a cada cliente o pedido una vez finalizado.
- Comprobación y gestión de los equipos de frío que la compañía facilita a los clientes. Con esta función además de realizar la solicitud del préstamo de un equipo de frío, en las sucesivas visitas se utilizará para comprobar que los productos que se están almacenando no son de la competencia y que el equipo de frío se encuentra en buen estado.

- Geo localización de coordenadas del cliente.
- Generación de un informe diario para que el preventa pueda conocer la productividad del día y un resumen de productos vendidos por cliente.

Todas estas funcionalidades están implementadas y almacenadas en el terminal, de forma que cuando comunique con el servidor pueda enviar toda la información recogida durante el día.

4.5.1 Autenticación de usuario

Para garantizar el acceso seguro a la aplicación es necesario realizar una autenticación de usuario y contraseña. Este código de usuario estará almacenado en la tabla *user* de la base de datos.

Ilustración 28. Pantalla de acceso

En caso que el usuario introduzca mal su usuario y contraseña se le mostrará el siguiente mensaje indicando que se ha introducido una autenticación errónea.

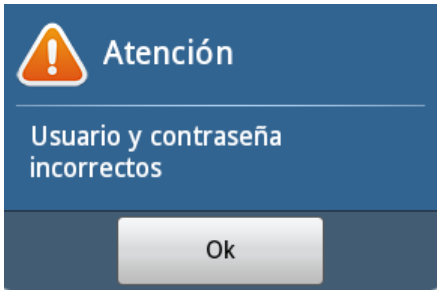


Ilustración 29. Mensaje de autenticación incorrecta

Si se produjese el caso que el usuario introdujese el usuario y la contraseña de forma errónea diez veces la aplicación se bloquearía y sería necesario realizar una carga inicial de datos en la base de datos que volviera a poner el contador del número de autenticaciones erróneas a cero. Dicho contador se actualizará a valor cero cada vez que se la autenticación en la aplicación sea satisfactoria.

A partir de la séptima vez que se produzca una autenticación errónea empezara a parecer el siguiente mensaje.

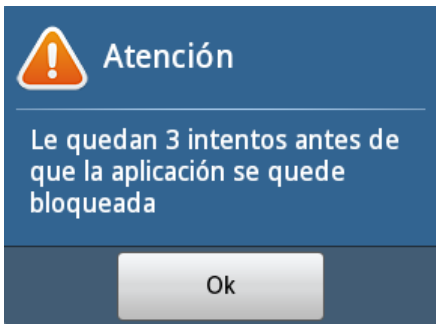


Ilustración 30. Mensaje de intentos restantes

Si la aplicación llegara a bloquearse este sería le mensaje que se le mostraría al usuario.

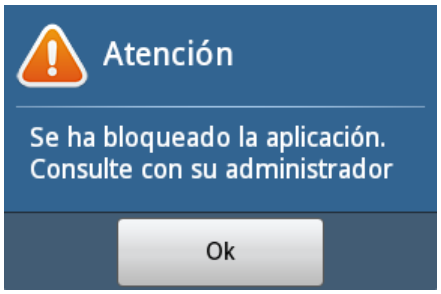


Ilustración 31. Mensaje de bloqueo de la aplicación

4.5.2 Menú principal

Una vez autenticado el usuario tendrá acceso a la pantalla principal de la aplicación en la que se lista a los clientes planificados para visitar en el día de la semana en que se encuentre. Desde la pantalla principal se tendrá acceso al menú principal de la aplicación desde el que se pueden acceder a las diferentes funcionalidades del operativo.

Detallaremos todas las funcionalidades del menú principal en los siguientes apartados.

Ilustración 32. Menú principal

- **Productos**: opción que nos permite consultar los productos disponibles.
- **Informes**: opción para consultar los informes del día.
- **Comunicaciones**: opción para comenzar la comunicación con le servidor.
- **Exportación**: opción para generar un fichero de exportación, en caso que no funcionen las comunicaciones nos permitiría enviar por otro medio el fichero al servidor para poder incorporar los pedidos al negocio.
- **Más**: información y configuración de la aplicación.
- **Salir**: opción para cerrar la aplicación.

4.5.3 Clientes

Como ya se ha comentado esta sería la pantalla principal de la aplicación. Por defecto, una vez el usuario se autentique se dirigirá a la pantalla con el listado de clientes planeados para visitar en día. La lista deslizable ofrece un listado con todos los clientes mostrando el nombre y el estado de la visita en caso de que ya se haya realizado.

Para seleccionar un cliente de la pantalla principal se deberá pulsar encima del cliente seleccionado y un submenú emergente aparecerá con todas las acciones que se puede realizar sobre el cliente.

En la parte superior izquierda de la pantalla, esta disponible el icono que permite expandir y contraer el panel de búsqueda.

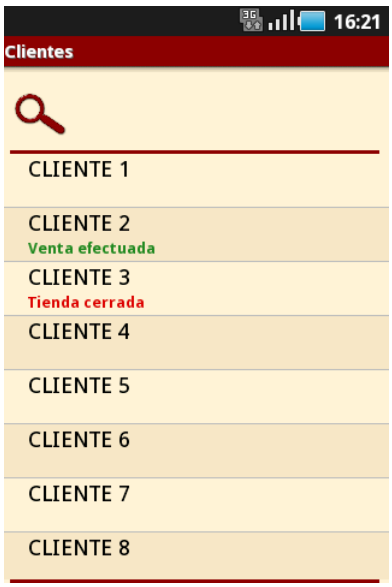


Ilustración 33. Listado de clientes

4.5.3.1 Búsqueda de clientes por código o por ruta

Una vez se despliega el panel de búsqueda pulsando sobre el icono de la lupa de la parte superior izquierda, el combo de selección de rutas y la caja de texto para buscar por código o por nombre estarán disponibles en la pantalla.

La búsqueda por rutas permitirá buscar, los clientes programados para el día actual, todos los clientes de la base de datos o los clientes asociados a las diferentes rutas del usuario En la parte inferior del panel de búsqueda esta disponible el campo de texto libre, en el que su puede realizar una búsqueda por código o por nombre de cliente. Cuando se pulse sobre el campo de texto el teclado emergente aparecerá.

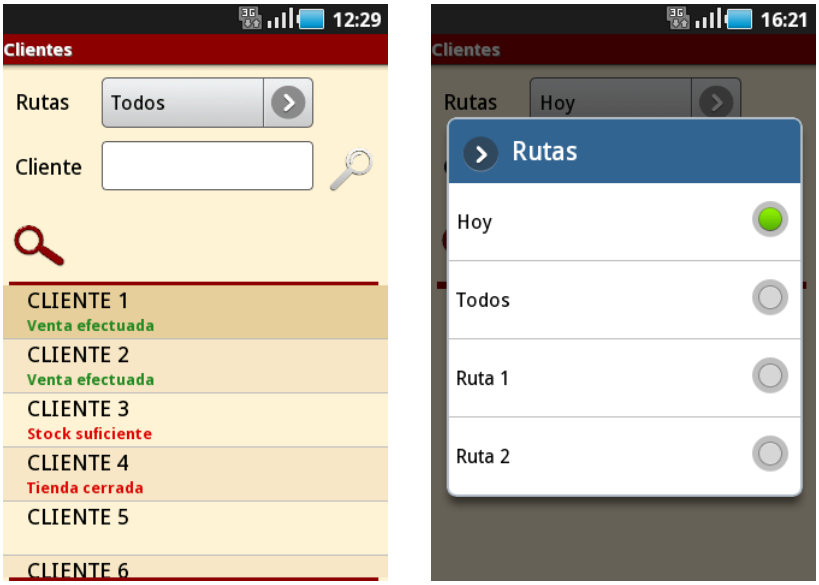


Ilustración 34. Panel de búsqueda de clientes

4.5.3.2 Selección de clientes

Para seleccionar un cliente se ha de pulsar encima de la línea de la lista asociada a la cuenta y el siguiente menú emergente aparecerá.

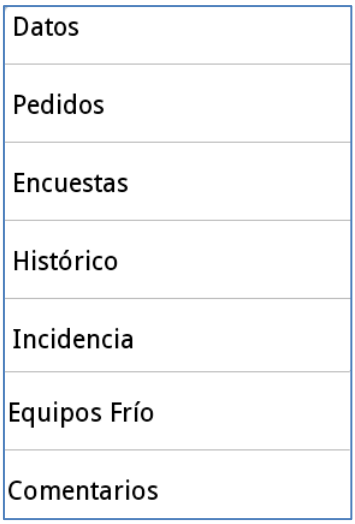


Ilustración 35. Menú emergente sobre los clientes

- **Datos**: opción que permite consultar y modificar los datos de un cliente.
- **Pedidos**: opción que permite crear y modificar los pedidos de un cliente.
- **Histórico:** opción que permite mostrar los últimos seis pedidos realizados por un cliente
- **Incidencias**: opción que permite tomar nota de la razón por la que un cliente no realiza ningún tipo de pedido.
- **Encuestas**: opción que permite realizar las encuestas enviadas desde central a un cliente.
- **Equipos de frío**: opción que permite tramitar la solicitud o chequear el correcto estado de un equipo de frío.
- **Comentarios**: opción que permite introducir un mensaje o comentario asociado a un cliente.

4.5.3.3 Datos del cliente

Los datos de cada cliente se muestran en las siguientes cuatro pantallas en las que se recogen los datos.

Ilustración 36. Pantallas 1 y 2 del detalle de clientes

Los campos del cliente que se definen en las dos primeras pantallas son:

- Código
- Nombre
- Dirección
- Ciudad
- Teléfono
- Tipo
- Canal
- Categoría
- Latitud y longitud
- Latitud y longitud capturadas desde el dispositivo.

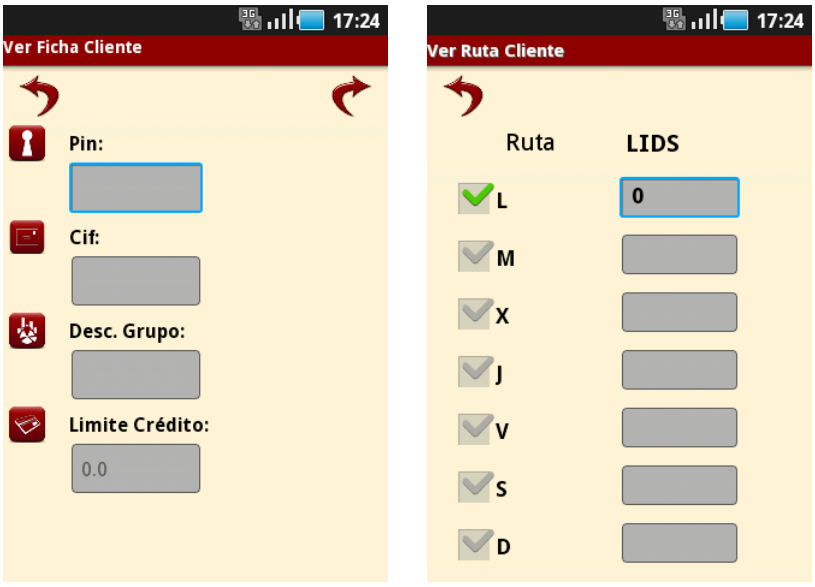


Ilustración 37. Pantallas 3 y 4 del detalle de clientes

- Pin
- Cif
- Grupo de descuento
- Límite de crédito
- Nombre de la ruta
- Día de la semana en la que el cliente esta planeado para visitarse y la hora a la que se estima que se realiza la visita, de lunes a domingo

4.5.3.4 Modificación de clientes

Además de consultar los datos del cliente se pueden modificar, para ello desde botón de menú del dispositivo se lanza el siguiente menú con las opciones modificar y grabar.

Es posible modificar todos los datos del menos el código del cliente.

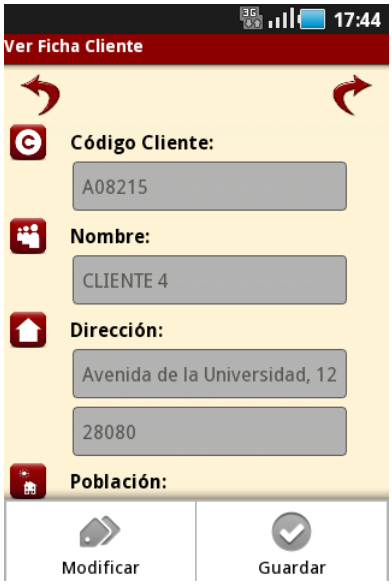


Ilustración 38. Menú de clientes

4.5.4 Pedidos

Esta opción disponible desde el submenú de clientes, permite al usuario pre-venta llevar a cabo la visita. Al hacer *click* sobre el cliente seleccionado, el siguiente submenú aparecerá. Las opciones de ***Pedidos*** e ***Incidencias*** permiten tomar nota de la vista realizada al cliente.

Ilustración 39. Menú emergente con las opciones del cliente

4.5.4.1 Insertar una incidencia

Si se pulsa sobre la opción incidencia, se podrá tomar nota de porque no se realizan ventas en el cliente seleccionado. Una vez tomada nota de la incidencia en el listado de clientes de la pantalla principal se podrá ver la razón en color rojo por la que no se han realizados ventas en el cliente.

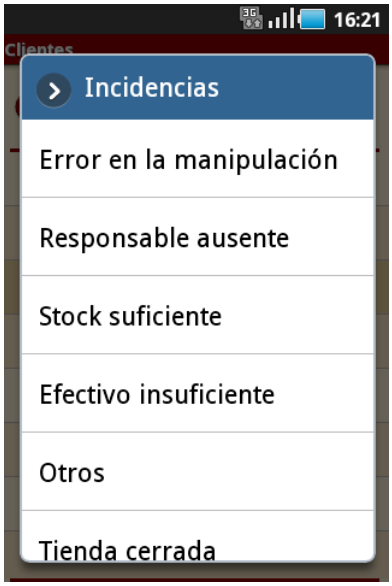


Ilustración 40. Posibles incidencias de la visita

4.5.4.2 Insertar un nuevo pedido

Para insertar un nuevo pedido, se ha de seleccionar la opción ***Pedido*** sobre el submenú de clientes. En la pantalla de principal de pedidos se listarán los pedidos

realizados y que aun no han sido enviados al servidor para ser tratados. Se pueden añadir nuevos pedidos pulsando sobre el icono situado en la parte superior derecha de la lista.

En esta pantalla se podrá observar la siguiente información:

- Información del cliente (nombre y código).
- Código del pedido.
  - Importe total del pedido.
  - Descuesto total aplicado en el pedido.
  - Estado del pedido en caso de estar confirmado.
  -

Ilustración 41. Listado de pedidos de un cliente

Añadir productos al pedido

Cuando se va a crear un Nuevo pedido la aplicación nos ofrece un listado con todos los productos disponibles para ser vendidos. En el listado de productos se pueden ver el número de cajas vendidas para este pedido, el precio de cada caja.

Ilustración 42. Detalle del pedido

Se pueden filtrar los productos de la lista por familia, tamaño, sabor o buscando por la cadena de código o descripción. Pulsando sobre el icono de la parte superior izquierda de la pantalla el panel de búsqueda se mostrará u ocultará.

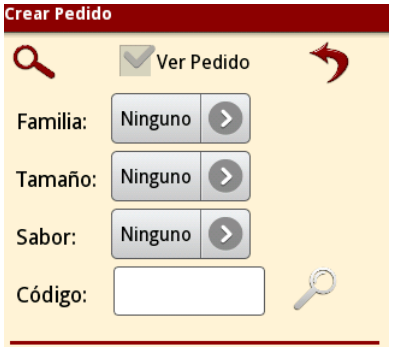


Ilustración 43. Panel de búsqueda de productos

Se selecciona un producto pulsando sobre él, de forma que el diálogo que permite introducir la cantidad de cajas vendidas aparece en la pantalla.

Para añadir un producto al pedido, se debe añadir la cantidad de cajas del producto vendidas y así se irán añadiendo sucesivamente al pedido.

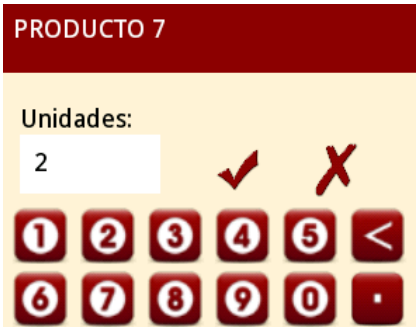


Ilustración 44. Diálogo de introducción de cantidades

Una vez cerrado el diálogo desde el que se añaden las cantidades, la cantidad de cajas vendidas aparecerá asociada al producto en la lista de productos disponibles.

Ilustración 45. Inserción de un producto al pedido

Aplicación de promociones

Cuando se añade un producto al pedido es posible que tenga promociones asociadas. Las promociones están compuestas por:

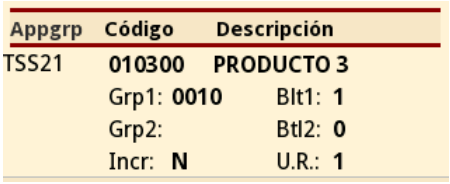


Ilustración 46. Detalle de una promoción

Cuando se añade un producto al pedido, la aplicación se encarga de chequear si hay alguna promoción asociada al producto. Si hay alguna promoción asociada al producto para el cliente sobre el que se está realizando el pedido, la promoción se aplicará directamente, si hubiera mas de una promoción disponible se ofrecería una pantalla emergente al usuario para que eligiera que promoción es la que quiere aplicar.

Una vez se ha seleccionado la promoción, la aplicación se encarga de recoger el ***Grupo de Regalo 1*** y de calcular los posibles productos regalos asociados a ese grupo y la cantidad de productos que se va a regalar, en caso de que haya varios posibles productos regalos una lista emergente aparecerá para que el usuario decida que producto es el que quiere regalar. Si solo hubiese un único producto para regalar, se añadiría

directamente. El mismo proceso se realizaría si la promoción tuviese ***Grupo de Regalo 2.***

Ilustración 47. Diálogos de elección de promoción y regalos

Cuando el producto regalo se ha añadido al pedido, se podrá ver desde el listado de productos referenciado por una letra P.

Ilustración 48. Aplicación de una promoción

Cálculo del pedido

Mientras se está tomando nota del pedido el, pre- venta podrá calcular el total de pedido que se lleva acumulado hasta el momento sin llegar a confírmalo. La casilla de ***Calculadora*** permite aplicar los descuentos, las promociones y los recargos por entrega o por alquiler, y así poder ver el total acumulado del pedido antes de llegar a confirmarlo.

Antes de confirmar el pedido los datos de total bruto, neto, descuentos y cajas vendidas parecerán en rojo para indicar que el pedido aun no ha sido confirmado.

Ilustración 49. Pantallas del cálculo del total pedido

Cuando se pulsa sobre el botón de ***Calculadora*** también se mostrarán las líneas de pedido asociadas a precio de Envío (E), precio de Alquiler (A) y Promociones (P).

Las líneas de pedido de Envío harán referencia a aquellos productos que tiene asociados un precio por llevar a cabo la entrega, el precio del envío será proporcional a las cajas de producto solicitadas.

Las líneas de pedido de Alquiler harán referencia a aquellos productos que tienen asociado un precio de alquiler (del envase del producto).

Las líneas de pedido de las Promociones harán referencia a los productos de regalo asociados al pedido.

Confirmación del pedido

Cuando el pedido se confirma, la fecha y la hora se registran en la base de datos. Una vez confirmado el pedido la aplicación volverá al listado de pedidos asociados al cliente seleccionado. Los pedidos confirmados solo podrán ser consultados o borrados, pero no modificados.

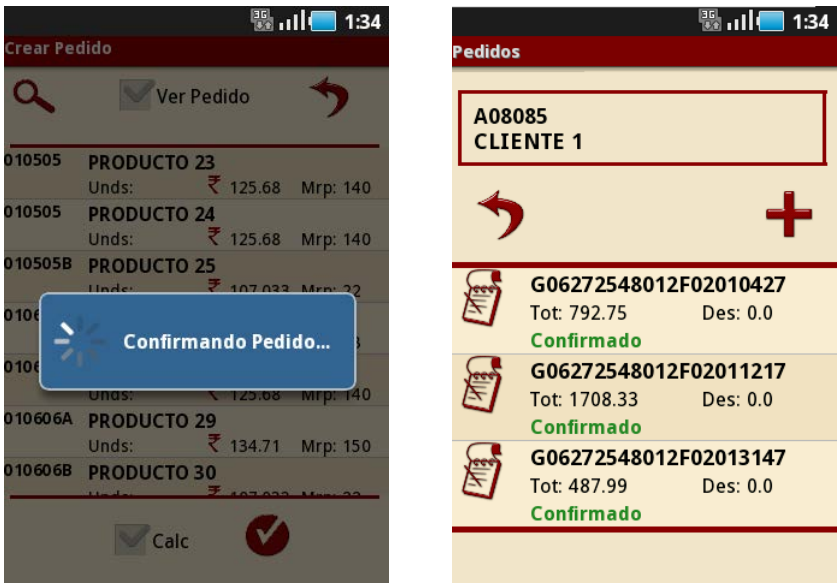


Ilustración 50. Confirmación de un pedido

Promociones y descuentos

Desde la pantalla de creación del pedido pulsando sobre el botón del menú aparecerán las opciones ***Promociones*** y ***Descuentos***. Estas dos opciones son sólo de consulta y permiten conocer todas las promociones y descuentos asociados al cliente seleccionado y que podrán aplicarse al dar de alta un pedido.

Ilustración 51. Menú de detalle del pedido

Ilustración 52. Lista de Promociones y Descuentos de un cliente

Modificación y borrado de pedidos

Una vez confirmado un pedido, sólo podrá ser consultado o borrado pero no modificado.

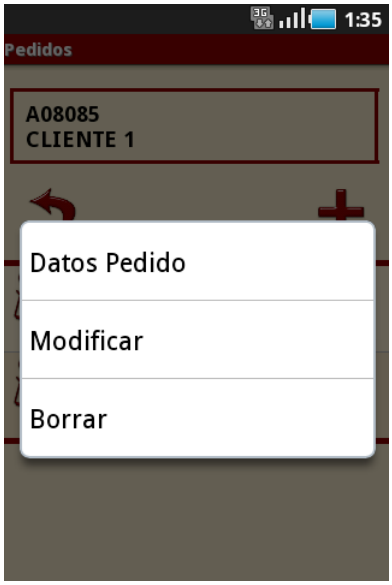


Ilustración 53. Acciones sobre un pedido

Si se trata de modificar un pedido ya confirmado aparecerá este dialogo:

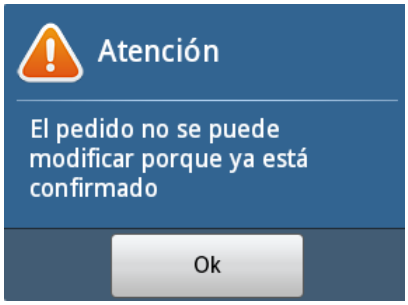


Ilustración 54. Diálogo de no modificación

Todos los pedidos pueden ser borrados, tanto los confirmados como los no confirmados, pero siempre antes de comunicar y enviar los datos al servidor. Si se quiere borrar un pedido el siguiente dialogo de confirmación aparecerá:

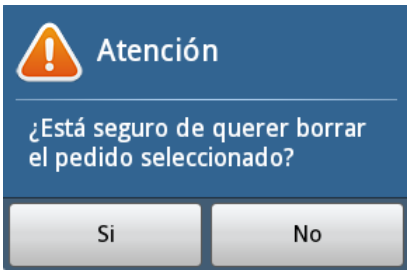


Ilustración 55. Diálogo de borrado de un pedido

4.5.5 Comentarios

Esta opción permite introducir diferentes tipos de comentarios asociados a cada cliente. El usuario ha de seleccionar un tipo de cometario y posteriormente podrá rellenar el campo de texto libre para tomar nota del comentario.

Ilustración 56. Selección del Comentario

En el detalle del mensaje el usuario puede introducir el comentario sobre e cliente.

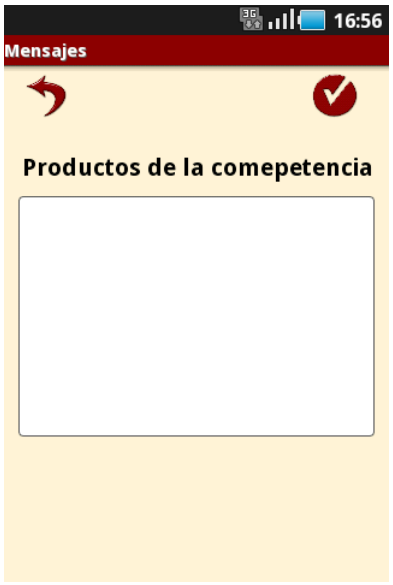


Ilustración 57. Pantalla de inserción de comentarios

4.5.6 Encuestas

Desde el menú emergente de Clientes, el usuario puede seleccionar la opción de ***Encuestas***. Esta opción permite consultar y rellenar las encuestas del cliente.

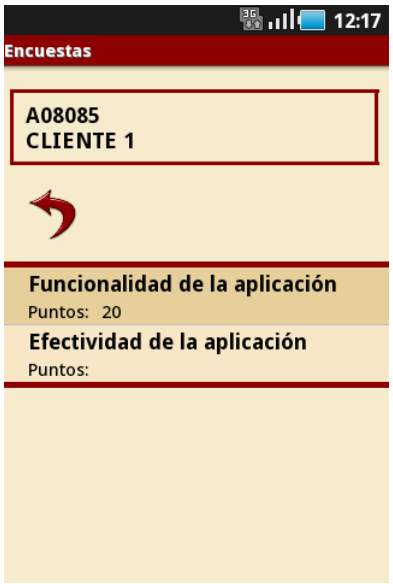


Ilustración 58. Listado de encuestas de un cliente

Para elegir una encuesta, se ha de hacer click para seleccionarla sobre la lista de posibles encuestas asociadas al cliente y las preguntas con las posibles respuestas irán apareciendo sucesivamente hasta finalizar la encuesta.

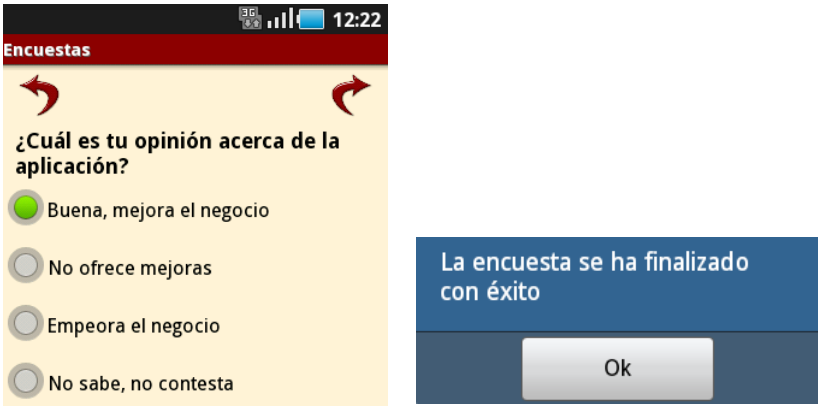


Ilustración 59. Preguntas y mensaje de finalización de una encuesta

4.5.7 Histórico

La aplicación mantiene un histórico con los 6 últimos pedidos asociados a cada cliente.

4.5.7.1 Histórico de clientes

En el menú emergente de ***Clientes*** el usuario puede seleccionar la opción de ***Histórico*** en la que se permite consultar los últimos pedidos realizados y enviados al servidor. El histórico del pedido se va actualizando automáticamente cada vez que se comunica con el servidor, insertando las nuevas órdenes de pedido y borrando las más antiguas. Con la configuración actual se van a mantener en el dispositivo los seis últimos pedidos de cada cliente.

En la lista de los históricos de un cliente se puede ver el código del pedido, la fecha en la que se realizó, el neto total y si se ha entregado o no con éxito. Haciendo click sobre cada pedido se puede ver el detalle del mismo y las líneas de pedido que tiene asociadas.

Ilustración 60. Listado del histórico de un pedido

4.5.7.2 Detalle del histórico

En la ventana del detalle del histórico aparecerán la lista de líneas y detalles que forman el pedido. En la lista hay datos relativos al pedido seleccionado: como el total de cajas solicitadas, el total de cajas entregadas y la razón por la que no han podido ser entregadas, en caso que no se hayan podido entregar todo el pedido. Los datos históricos sólo pueden ser visualizados y no se pueden modificar.

Ilustración 61. Detalle del histórico de un pedido

4.5.8 Equipos de frío

Esta opción esta disponible desde el menú emergente de ***Clientes*** y permite al usuario pre-venta dar de alta nuevos equipos de frío o chequear equipos ya existentes.

Ilustración 62. Listado de equipos de frío de un cliente

4.5.8.1 Localización del cliente

El usuario pre-venta podría recoger las coordenadas GPS del cliente a través de la aplicación, para verificar la ubicación de los equipos de frío prestados al cliente. La localización de los clientes se almacenará en la base de datos en la tabla de clientes. Según se acceda al listado de equipos de frío de un cliente, se lanza la captura de las coordenadas.

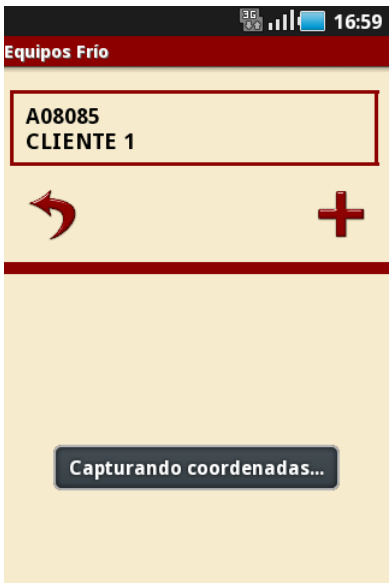


Ilustración 63. Captura de coordenadas

Cuando la aplicación detecta las coordenadas del cliente mostrará el siguiente mensaje al usuario para confirmar si se desean almacenar.

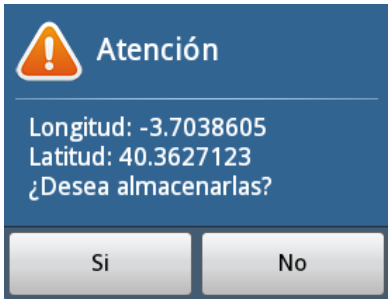


Ilustración 64. Diálogo de captura de coordenadas

4.5.8.2 Detalle del equipo de frío

El usuario podrá añadir un nuevo equipo de frío o chequear uno existente, tomando nota del detalle del mismo. Cada cliente podrá tener uno o más equipos de frío asociados, y deberán de chequearse por separado.

Si el usuario solicita un nuevo equipo de frío para el cliente, la operación marcada por defecto será ***Solicitud*** y no podrá modificarse mientras se esté dando de alta.

Ilustración 65. Detalle del Equipo de Frío

Se podrán tomar fotos y capturar el código QR de cada equipo de frío. El código QR capturado se convertirá en el ID único asociado al equipo.

Ilustración 66. Opciones de menú del Equipo de Frío

4.5.8.3 Captura de fotos

Pulsando sobre la opción ***Fotos*** del menú el siguiente menú emergente aparecerá para seleccionar si se quiere tomar una foto o visualizar las fotos ya tomadas asociadas a este equipo de frío.

Ilustración 67. Opciones de menú Foto

4.5.8.4 Escaneo de códigos QR

Si el usuario selecciona la opción de menú ***Escanear*** aparecerá la pantalla de escaneo de códigos. Sólo se permite el escaneo de un código por equipo de frío, por lo que siempre prevalecerá el último código capturado.

La pantalla de captura de los códigos es la siguiente

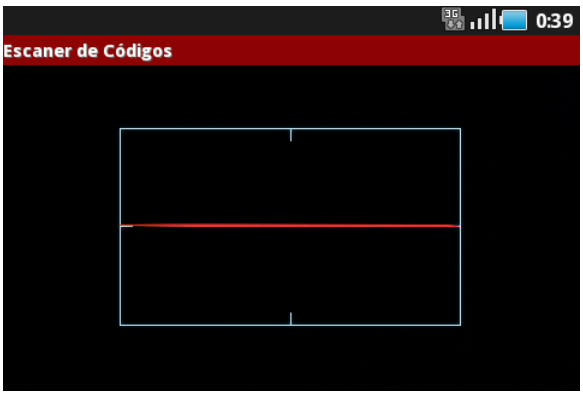


Ilustración 68. Escáner de Códigos

4.5.9 Productos

Esta opción disponible desde el menú principal de la aplicación, permite listar todos los productos disponibles.

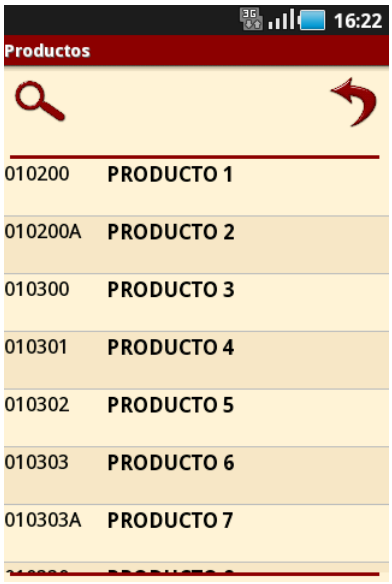


Ilustración 69. Listado de productos

En la parte superior de la pantalla se puede desplegar el panel de búsqueda.

Ilustración 70. Panel de búsqueda de productos

En el panel de búsqueda se puede buscar por los combos de ***Familia***, ***Tamaño*** o ***Sabor***, o por campo de texto libre para buscar por código o por descripción.

4.5.9.1 Detalle del producto

Pulsando sobre uno de los productos de la lista se puede ver el detalle del producto seleccionado. De acuerdo con el escenario propuesto se ha decidido tomar como producto de una caja de refrescos. En la pantalla del detalle del producto se pueden ver las características del producto código, número de botellas por caja, número de serie, familia, tamaño, sabor y producto terminado.

Ilustración 71. Detalle del producto

4.5.10 Informes

La aplicación ofrece una serie de informes del día actual, para resumir las ventas realizadas por el usuario. Desde el menú principal de la aplicación se puede acceder a los informes.

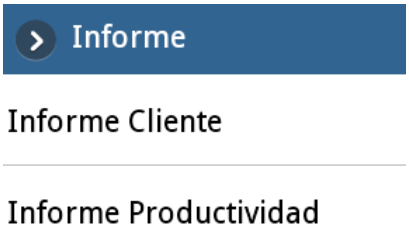


Ilustración 72. Menú emergente de Informes

4.5.10.1 Informe por clientes

Esta opción está disponible desde el submenú de informes. El usuario puede obtener un resumen de los clientes visitados al final del día. En la parte superior de la pantalla se puede ver la fecha del día, las visitas planeadas para el día actual, las visitas realizadas, la proporción de casos de éxito (visitas con pedidos/visitas planeadas) y el total que ha sido vendido en el día.

La lista muestra las cajas y el neto vendido a cada cliente, si el cliente ha sido visitado con incidencia entonces aparecerá la incidencia registrada.

Ilustración 73. Informe de clientes

En la lista del informe la información que aparece relacionada con cada cliente es: nombre del cliente, número de cajas vendidas, total de neto vendido para este cliente, incidencia en caso de haberla

Detalle del informe por cliente

Al pulsar sobre cada cliente el usuario pre-venta podrá revisar el informe del día asociado a ese cliente. En la pantalla de detalle se puede ver el total de caja entregadas en el día y el porcentaje total de entrega.

Ilustración 74. Detalle del informe del cliente

4.5.10.2 Informe de productividad

Esta opción también disponible desde la opción de ***Informes*** del menú principal, permite al usuario pre-vente tener una visión de la productividad del día de trabajo realizado. Esta opción muestra la cantidad de visitas productivas realizadas.

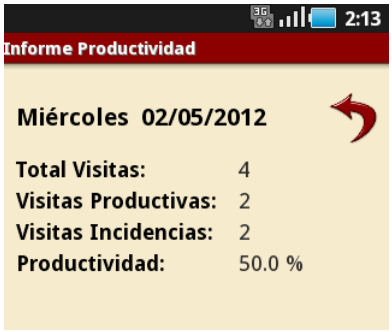


Ilustración 75. Informe de productividad

4.5.11 Exportación

La ***Exportación*** esta disponible desde el menú principal. Esta opción permite crear el fichero de exportación en el directorio de la tarjeta externa del fichero por si hubiera problemas al comunicar desde el dispositivo al servidor y se quisiese enviar el fichero por un medio externo. El fichero se almacenará en el directorio configurado desde las opciones de ***Configuración***.

Se recomienda tener una tarjeta de memoria externa en el dispositivo, para generar el directorio en el que almacenar los ficheros de logs. Si el teléfono no dispone de tarjeta de memoria se recomienda habilitar la opción de "*Permitir ubicaciones falsas*" que montar un sistema de ficheros emulando la tarjeta de memoria.

4.5.12 Configuración

La ***Configuración*** está disponible desde la opción ***Más*** del menú principal de la aplicación. Esta opción permite configurar las opciones de la aplicación y de las comunicaciones.

Es necesario autenticarse para poder acceder a la configuración. La contraseña esta almacenada en la Base de Datos de la aplicación, en la tabla *Parameters* en el campo *PaswordPreferences*.

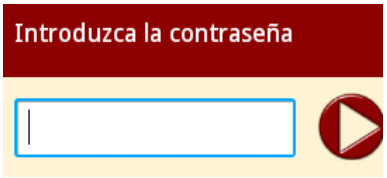


Ilustración 76. Autenticación de la configuración

Una vez se autentica el usuario, aparece la pantalla principal de la configuración. En esta pantalla se podrá configurar el buzón del usuario con el que comunicará el dispositivo, de forma que el servidor sepa identificarlo y los *timeouts* de conexión.

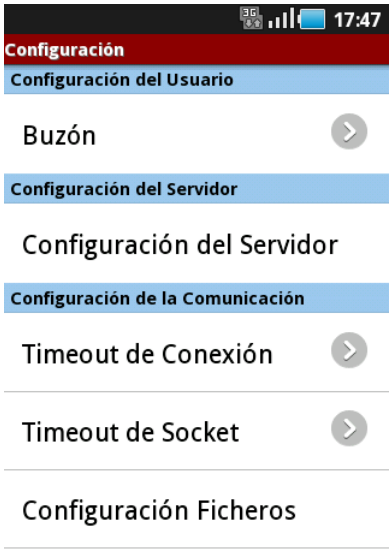


Ilustración 77. Pantalla principal de la Configuración

Desde esta pantalla se podrá acceder a las pantallas secundarias de configuración del servidor y de ficheros.

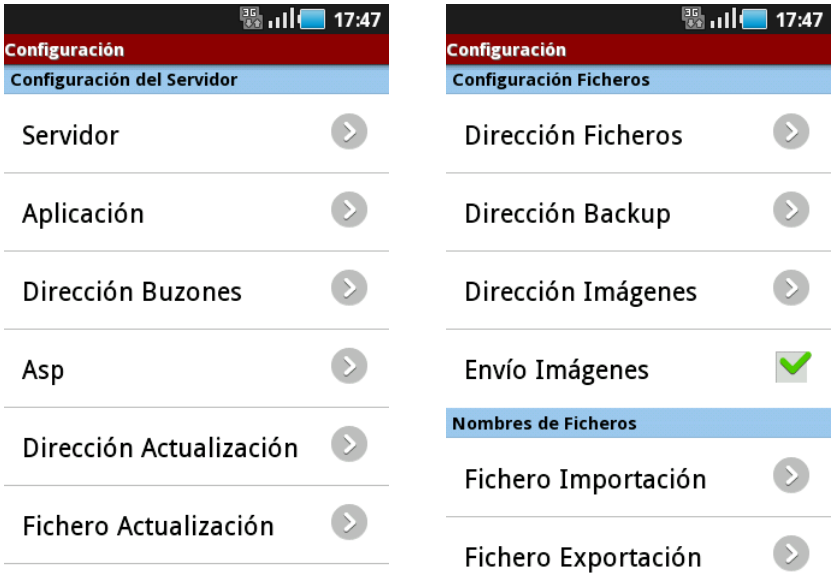


Ilustración 78. Configuración del servidor y de ficheros

4.5.13 Comunicaciones

La opción de ***Comunicaciones*** está disponible desde el menú principal de la aplicación. Esta opción se encarga de comunicar con el servidor de la aplicación envía los datos recogidos por el dispositivo y se descarga las actualizaciones de datos en caso de que haya datos que descargar asociados al usuario y se encarga de importarlos en la Base de Datos.

La aplicación esta preparada para cargar los datos descargados en la base de datos, de forma que sabe distinguir si los datos que se están importando se deben insertar directamente en la base de datos o si por el contrario son actualizaciones de datos ya existentes, en cuyo caso deberán actualizar los registros.

El proceso de comunicación esta formado por los siguientes pasos:

- Exportación de los datos recogidos por el usuario en un fichero de texto.
- Envío de los datos al servidor.
- Chequeo del servidor para comprobar si hay algún fichero que descargar.
- Importación de los datos en la Base de Datos en caso de haberlos.
- Actualización de la Base de datos para indicar que los datos se han enviado correctamente.

Ilustración 79. Diálogos del proceso de comunicación

4.5.13.1 Testeo de comunicaciones

El ***Test*** está disponible desde la opción ***Más*** del menú principal. Esta opción permite testear la comunicación con el servidor, sin llegar a realizar el proceso de comunicación completo.

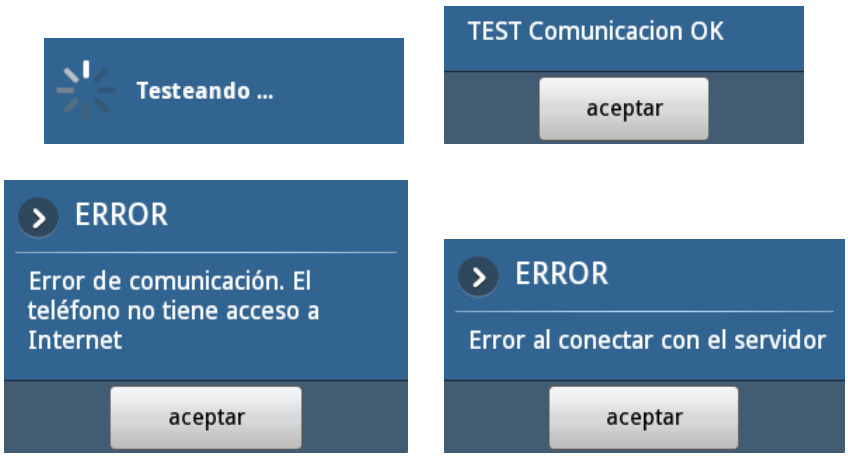


Ilustración 80. Diálogos del proceso de Test

4.5.14 Actualización de la aplicación

Cuando una nueva versión de la aplicación esté disponible, se ha implementado una forma de actualización automática, para que la nueva versión se descargue y se actualice en el dispositivo.

La gestión de versiones se va a llevar a cabo en la Base de Datos, de forma que en las comunicaciones con el servidor, se irá informando a los dispositivos de la última versión de la aplicación disponible.

Esta comprobación se hace cada vez que se inicia la aplicación en los dispositivos, si detecta que la última versión en posterior a la actual, aparecerá el mensaje indicando que hay una nueva versión para descargarse.

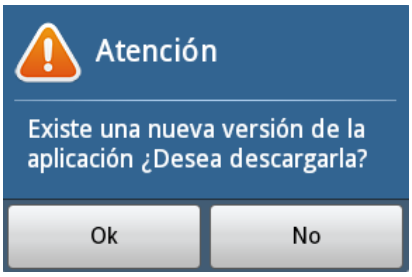


Ilustración 81. Diálogo de información de nueva versión

En caso de que el usuario acepte, el fichero se descargará. Una vez haya finalizado la descarga se iniciará la instalación del fichero *.apk*. El dispositivo avisará que la aplicación va a ser remplazada por una versión superior.

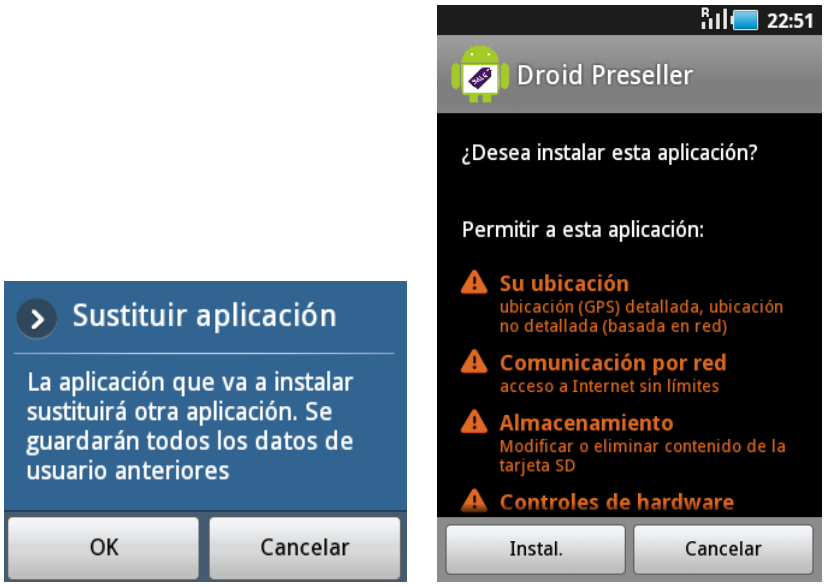


Ilustración 82. Mensajes de actualización de una aplicación

Una vez se acepte la aplicación se actualizará pero la Base de datos no se remplazará.

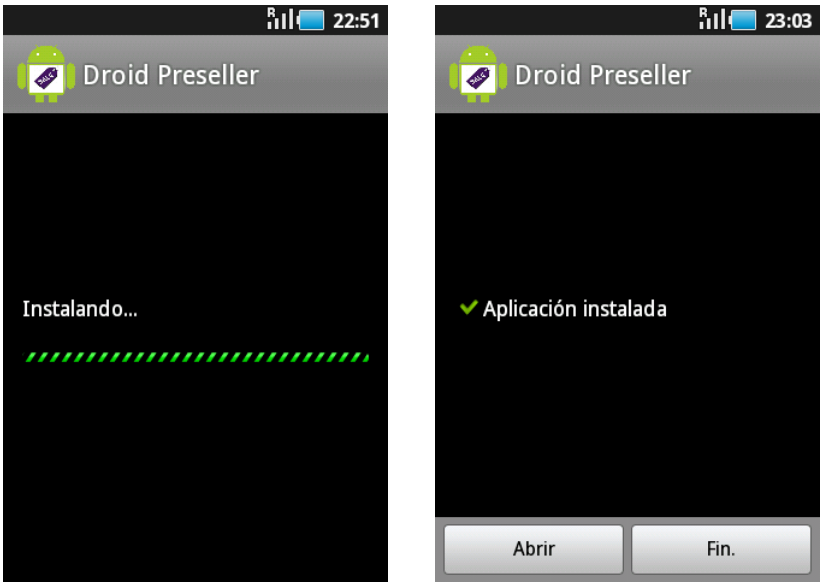


Ilustración 83. Mensajes de instalación

4.6 Pruebas

En este capítulo se explicará el plan de pruebas de la aplicación, el diseño de las mismas así como los resultados obtenidos.

Como ya se ha comentado anteriormente, la aplicación utiliza una **versión de Android 2.2 (API nivel 8).** Todas las pruebas se han realizado tanto en el emulador facilitado por el *SDK de Android* como en dispositivos reales (*Samsug Galaxy ACE versión 2.2.1* o *Samsung Galaxy SII versión 2.3.3*) de diferentes versiones para garantizar la compatibilidad de aplicación.

4.6.1 Pruebas de negocio

Batería de pruebas orientada a comprobar las funcionalidades de la aplicación.

- **Administración de pedidos.** Se realizan pruebas de generación de pedidos con y sin promociones y con y sin descuentos. Estos pedidos han de poderse modificar antes de darlos por confirmados, y han de poderse borrar siempre antes de ser enviados. Todas las pruebas se realizan satisfactoriamente.

- **Captura de coordenadas.** Se prueba a capturar las coordenadas a través de la aplicación con y sin la señal de GPS conectada. En ambos casos el resultado de la prueba es satisfactorio ya que la señal puede capturarse por el receptor de GPS o través de los datos de red.

- **Captura de códigos QR y códigos de barra.** Se realizan diferentes capturas de códigos QR y códigos de barras de diferentes tamaños. Las pruebas son satisfactorias.

- **Captura y visualización de fotos.** Se realizan pruebas de capturas y visualización de fotografías a través de la aplicación. Las pruebas son satisfactorias.

- **Modificación de clientes.** Se realizan pruebas de modificación de clientes, y se comprueba que los datos son correctamente visualizados y almacenados en la base de datos.

- **Realización de encuestas.** Se realizan pruebas de diferentes encuestas asociadas a diferentes clientes y se comprueba que los datos se han almacenados correctamente en la Base de Datos y se permite su posterior visualización.

- **Generación y visualización de históricos.** Se realizan diferentes pedidos y después de enviarlos, se comprueba como han sido borrados de las tablas de

pedidos y almacenados en las tablas de históricos. También se realizan pruebas para comprobar el mantenimiento de las tablas de históricos para ir almacenando solamente los seis últimos pedido de cada cliente y de ese modo que la tabla no crezca de forma infinita. Las pruebas se realizan con éxito.

4.6.2 Pruebas de interfaz

Batería de pruebas orientadas a comprobar la correcta visualización de las pantallas que componen la aplicación así como su configuración.

- **Multidioma de la aplicación.** Se realiza satisfactoriamente el cambio de idioma de la aplicación al cambiar el idioma por defecto del teléfono.

- **Persistencia de los ficheros de configuración.** Se realizan con satisfacción cambios en los ficheros de configuración de la aplicación, que permanecen correctamente guardados después de cerrar la aplicación o apagar el teléfono.

- **Scroll de la listas.** Carga de gran cantidad de registros en la aplicación para comprobar si las listas se rellenan y desplazan correctamente.

- **Pruebas de navegación por la aplicación.** Se realizan pruebas de un uso completo de la aplicación interactuando con los diferentes módulos que la componen. Dejando la aplicación en segundo plano mientras se realizan o recepcionan llamadas.

4.6.3 Pruebas de instalación

- **Instalación de cero de la aplicación.** Se realizan pruebas de instalación de la aplicación a través de fichero .apk generado con la keystore correspondiente. La aplicación se instala correctamente y los permisos se aplican para su correcto funcionamiento.

- **Actualización de la aplicación.** Se realiza una actualización de la aplicación, copiando el nuevo fichero .apk en el dispositivo y descargándolo a través de la funcionalidad implementada. En ambos casos la aplicación se actualiza correctamente y los datos de la Base de Datos no se ven afectados por la actualización.

4.6.4 Pruebas de datos

- **Importación de datos.** Se copia directamente un fichero de importación en la ruta correspondiente del dispositivo y al abrir la aplicación la importación se realiza correctamente. La prueba se realiza satisfactoriamente.

- **Importación de datos con un fichero mal malformado.** Se realiza la misma prueba que la anteriormente descrita pero añadiendo errores de formato al fichero de importación. La aplicación escribe en el fichero de log correspondiente cual ha sido la razón de los errores en la importación. La prueba se realiza satisfactoriamente.

- **Exportación de datos.** Se realiza un nuevo pedido desde la aplicación y se prueba la exportación de los datos. La prueba se realiza correctamente ya que los datos asociados al pedido están en el fichero de exportación.

- **Proceso de comunicación completo.** Se realiza un proceso de comunicación completo, en el que se produce la exportación, se envían los datos, se descarga el fichero de importación, se importan los datos y se actualizan los estados. El proceso de realiza satisfactoriamente.

- **Proceso de comunicación con pérdida de conexión.** Se realiza un proceso de comunicación forzando fallos en la conexión en los diferentes estados del proceso para verificar que el proceso falla y no se actualizan los estados que sería la forma de indicar en la Base de Datos que los registros se han enviado correctamente y que se pueden borrar. Si el proceso fallase en la importación el fichero de importación no se borraría, por lo que se forzaría a importarlo antes de realizar la siguiente comunicación o la abrir la aplicación.

4.7 Resumen del proyecto

Con el objetivo de obtener una visión resumida del proyecto en términos de tiempos y costes, en este apartado se detalla la planificación final seguida durante el transcurso del proyecto, así como un presupuesto a modo de resumen de los costes del mismo.

4.7.1 Planificación

La duración en el tiempo del proyecto ha sido aproximadamente de 10 meses. Durante este periodo de tiempo, la dedicación al proyecto no ha sido uniforme, dedicándole durante algunas fases todo el tiempo posible mientras que en otras fases ha debido de compaginarse con otras actividades. Lo que significa un total de 880 horas de trabajo, tal y como se detallan a continuación.

Entre las tareas que se han llevado a cabo para la implementación del proyecto, cabría destacar tres grandes bloques:

- **Estudio de la plataforma Android.** En Septiembre de 2011, coincidiendo con el primer mes del desarrollo del proyecto, la tarea que se llevo a cabo fue la toma de contacto y aprendizaje de la plataforma Android. A pesar de que la autora del proyecto hubiera trabajado anteriormente con JAVA y con otras plataforma móviles como *c#* para desarrollos en Windows Mobile, fue necesario llevar a cabo un estudio para familiarizarse con la nueva plataforma.
  Además de la lectura y comprensión de la documentación acerca de Android, se llevó a cabo la instalación de las herramientas necesarias para el desarrollo de aplicaciones y la realización de una serie de ejemplos a modo de iniciación.
  El estudio de la documentación de Android ha sido una tarea viva a lo largo de todo el desarrollo del proyecto. Se estima que el tiempo empleado en esta tarea ha sido de unas 160 horas.

- **Desarrollo de la aplicación.** Entre los meses de Octubre de 2011 y Abril de 2012 se ha trabajado en la implementación de la aplicación. En la mayor parte de este periodo la dedicación al proyecto ha sido casi total, invirtiendo diariamente una típica jornada laborar de ocho horas en la mayoría de los días. El desarrollo de la aplicación puede dividirse en dos grandes bloques, un primer bloque desarrollado en primer lugar en el que se definieron e implementaron todas las pantallas y navegación a través de ellas de la aplicación. Y un segundo bloque en el que se le dio funcionalidad a la aplicación, llevando a cabo todo el desarrollo del negocio de la misma.
  Teniendo periodos vacacionales y otras tareas realizadas durante estos meses, se estima que la dedicación a esta tarea ha sido de 620 horas.

- **Redacción de la memoria.** Durante los meses de Marzo, Abril, Mayo y Junio de 2012 se ha realizado la documentación del presente documento. Donde se ha querido plasmar el trabajo realizado durante los pasados meses. Esta tarea se calcula que ha requerido una dedicación de 150 horas.

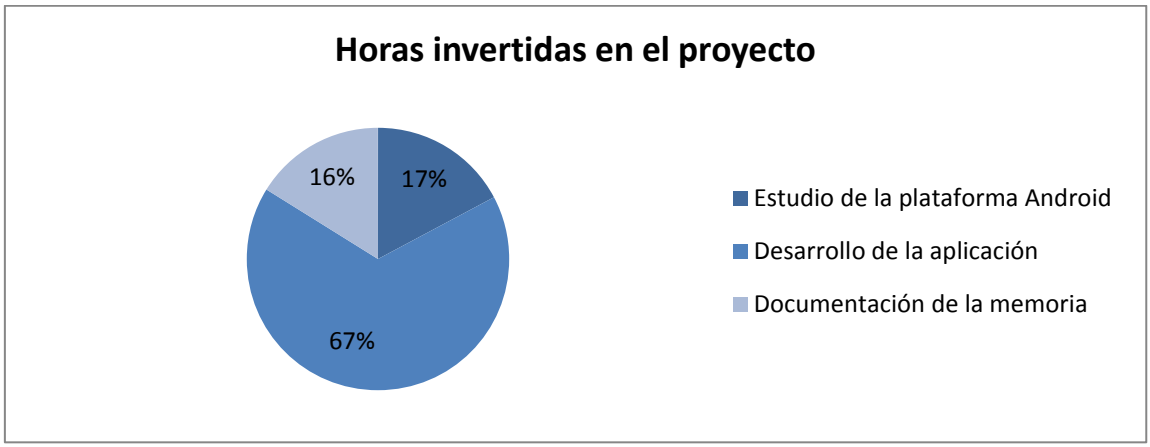


Tabla 18. Duración de las tareas del proyecto

En la siguiente figura, se ofrece una gráfica en la que puede comprobar el número de horas dedicas a cada una de las actividades del proyecto según el mes de desarrollo del mismo.

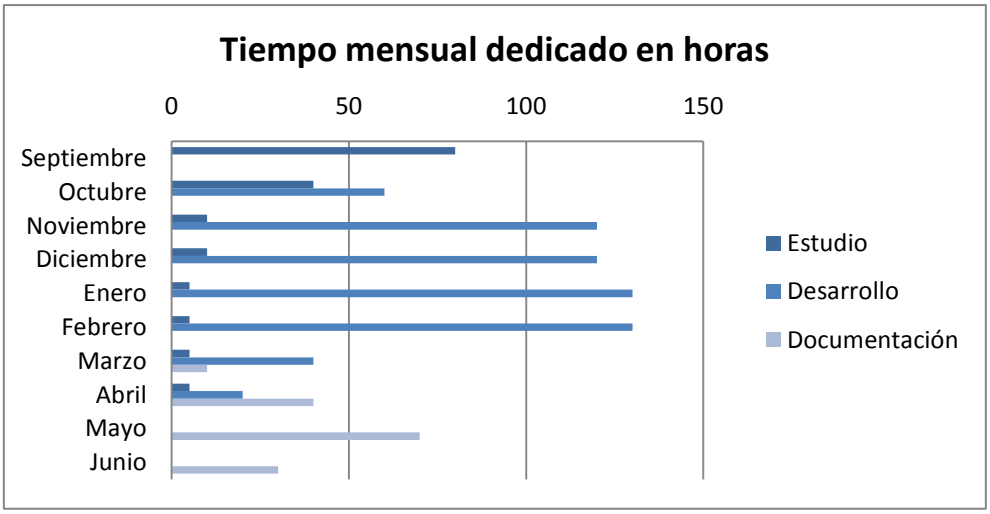


Tabla 19. Tiempo dedicado por meses y tareas

4.7.2 Presupuesto

A continuación se detalla el presupuesto final del proyecto. Los costes que conllevan el desarrollo del presente proyecto se han dividido en dos bloques coste de personal y coste de material e infraestructuras.

4.7.2.1 Presupuesto del personal

Para el desarrollo del proyecto han sido necesarios dos perfiles profesionales distintos, pero que a su vez se han complementado un analista y un programador. Debido a la finalidad del presente proyecto ambos perfiles han sido desarrollados una misma persona.

- **Analista:** encargado de estudiar Android, diseñar la aplicación y redactar la documentación más relevante, representada por dos tercios de la presente memoria.
- **Programador:** encargado de la implementación y pruebas de la aplicación, así como de la documentación de la misma.

Para llevar a cabo los cálculos presupuestados se han tenido en cuenta los sueldos medios estimados en España en los últimos meses (Infojobs, 2012). En la siguiente tabla se puede ver a relación entre los diferentes perfiles y las horas empleadas en el desarrollo del proyecto.

PRESUPUESTO EN PERSONAL					
	Estudio (h)	Desarrollo (h)	Documentación (h)	Salario (€/h)	Euros
Analista	100		60	14,20	2.272
Programador	60	620	110	12,78	10.096,2
TOTAL					**12.368,2**

Tabla 20. Presupuesto en personal

4.7.2.2 Presupuesto del material

Para el desarrollo del proyecto ha sido necesario el uso de material hardware, así como de una conexión a internet, imprescindible para la realización de las pruebas de comunicaciones. A continuación se detalla el material utilizado así como sus características:

- **Ordenador portátil Sony VAIO vpcsb,** procesador Intel(R) Core™ i5-2430 @ 2.40GHz, memoria RAM DDR de 4GB, disco duro de 500GB de capacidad entre otras características. (Sony, 2012)

- **Teléfono móvil Samsung Galaxy ACE,** procesador de 800MHz, pantalla de 3.5 pulgadas, cámara de 5 *megapixels* con flash LED, servicios Google, Wi-Fi, GPS. Plataforma Android 2.2 Froyo. (Smart-gsm.com, 2011)

- **Conexión ADSL,** ADSL Movistar a 6MB de velocidad.

En la siguiente tabla se puede ver el desglose total del material.

PRESUPUESTO EN HARDWARE E INFRAESTRUCTURA			
	Coste	Unidades	Euros
Portátil	930	1	930
Teléfono	60	1	225
Conexión	39,99(€/mes)	9	359,91
TOTAL			**1.514,91**

Tabla 21. Presupuesto en material e infraestructura

4.7.2.3 Presupuesto final

Resumiendo lo anteriormente expuesto acerca del presupuesto del proyecto, el resultado total se obtendrá de la suma del presupuesto en personal y el presupuesto en material e infraestructura utilizado. De esta forma el presupuesto total asciende a la cantidad de 13.244,11 euros. En la siguiente tabla se pude ver el resumen del presupuesto total del proyecto.

RESUMEN DE COSTES	
Tipo de Gasto	**Euros**
Gasto en personal	12.368,2
Gasto en material e infraestructura	1.514,91
TOTAL	**13.883,11**

Tabla 22. Presupuesto total del proyecto

Capítulo V. Conclusiones y desarrollos futuros

En este capítulo se hace un repaso global al proyecto, presentado las conclusiones finales en comparación con los objetivos marcados inicialmente y los posibles trabajos futuros.

5.1 Conclusiones

Una vez concluidas las principales tareas que forman este proyecto fin de carrera, es el momento en el que se puede hacer balance y crítica de los resultados obtenidos. Así pues, repasando los objetivos inicialmente marcados pueden sacarse las siguientes conclusiones:

- **Estudio las tecnologías móviles del mercado.**

  El estudio de las tecnologías móviles del mercado ha supuesto una parte mucho más interesante de lo que al principio se podía presuponer. El conocer la historia reciente de las tecnologías móviles que hoy en día tanto nos influyen, la velocidad a la que han evolucionado y el inminente cambio que ha supuesto en la sociedad, hacen que la finalidad de este objetivo inicialmente marcado haya superado sus expectativas con éxito. La comparativa llevada a cabo de las diferentes plataformas móviles ayudó sin lugar a duda a la decisión de elegir la plataforma Android como base sobre la que sustentar este proyecto.

- **Estudio del desarrollo de aplicaciones en Android.**

  A lo largo de la realización de este proyecto se ha conseguido obtener un conocimiento amplio de sistema operativo de Android. Su arquitectura, sus componentes y características, así como el funcionamiento y posibilidades ofrecidas se han ido conociendo gracias principalmente a la extensa documentación que Google ha puesto a disposición de los desarrolladores. Especialmente en las primeras fases del proyecto, llevando a cabo la toma de contacto con la tecnología, la ayuda de los manuales y documentación ha sido inestimable para poder llevar a cabo un acercamiento satisfactorio.

El objetivo que se marcó inicialmente del conocimiento de la plataforma se ha llevado a cabo realizando el estudio previo, la realización de los diferentes ejemplos y comprensión de manuales y documentación, así como durante la instalación de las herramientas necesarias para el desarrollo. La posterior documentación para la

realización del presente documento ha ayudado a afianzar todos los conocimientos adquiridos.

- **Diseño e implementación de la aplicación.**

El análisis y el diseño de la aplicación se han llevado a cabo pensando en las necesidades supuestas para el caso de uso tenido en cuenta en el desarrollo de este proyecto. Las dificultades encontradas durante el desarrollo de estos módulos vinieron principalmente relacionadas con la búsqueda de la usabilidad de la aplicación debido a la cantidad de módulos que se decidieron implementar. Fue por ello el diseño de la navegación por las diferentes pantallas un hito importante en el proyecto. El diseño de la base de datos y el mecanismo que permitiese hacer una correcta y eficiente importación y explotación de los datos trajo con sigo la segunda gran dificultad encontrada en estas fases.

Después de desarrollar los puntos anteriores se procedió a implementar una aplicación completa que cumpliera con las funcionalidades y requerimientos previamente diseñados.

El desarrollo de la aplicación podría considerarse la parte más laboriosa de todo el proyecto, pero a la vez se puede afirmar que ha sido la más entretenida de realizar. El hecho de que el lenguaje mayormente utilizado para el desarrollo haya sido JAVA mas XML, ha facilitado el hecho de trabajar con la plataforma. La gran comunidad de desarrolladores de Android ha facilitado y ayudado a solventar todas aquellas dificultades que se han ido encontrando a lo largo del desarrollo de la aplicación.

Se ha de destacar como parte de las dificultades encontradas a lo largo del desarrollo, la búsqueda en todo momento de un cierto atractivo visual. Al tratarse de una aplicación diseñada para dispositivos limitados, hay que tener en cuenta en todo momento la información que se decide mostrar y cómo mostrarla.

Como conclusión de este apartado se puede afirmar que la elección de Android para el desarrollo de la aplicación ha supuesto un acierto, dado que toda la funcionalidad que se ha decido implementar se ha llevado a acabo con éxito.

- **Pruebas de la funcionalidad de la aplicación.**

Las pruebas que se han llevado a cabo para verificar el buen funcionamiento de la aplicación han tenido resultados positivos en todos sus casos. Tal y como se explica en ese apartado se han realizado pruebas de funcionalidad del negocio, de interfaz, de instalación y de la correcta persistencia de los datos.

Se puede descarta como una dificultad encontrada a la hora de realizar todas la pruebas de comunicaciones, la necesidad que hubo de montar un servidor de

aplicaciones con el que se pudiese comunicar y verificar el correcto envío y recepción de los datos.

La instalación de la aplicación en distintos dispositivos reales y su correcto funcionamiento en ellos hacen que se puedan concluir como satisfactorias las pruebas realizadas sobre la aplicación.

- **Documentación del proyecto.**

La documentación del proyecto ha constituido una parte muy importante del desarrollo de proyecto, y de la que se puede afirmar ha ayudado a afianzar todos los conocimientos adquiridos durante el desarrollo del mismo.

Se puede concluir que Android, es una plataforma de lo más interesante para que un desarrollador plasme sus ideas. No sólo es atractiva a nivel de mercado debido al alto número de dispositivos a precios asequibles, sino que además intenta facilitar la tarea del desarrollador con el fin de aumentar el número de aplicaciones que pueden ofrecerse a sus usuarios.

5.2 Desarrollos futuros

En este capítulo se exponen algunas de las mejoras que podrían llevarse a cabo como futuras ampliaciones del proyecto.

La plataforma Android es todavía muy joven y, por tanto, está en continua evolución. Google ha diseñado un programa de actualización muy agresivo desde el comienzo de la plataforma, con importantes mejoras en el software y actualizaciones constantes en las librerías de desarrollo.

Una posible mejora sería la implementación del código sería el actualizarlo a la versión 4 de Android de forma que pudiera ser desplegada en móviles y tabletas.

La principal mejora que se podría implementar para el proyecto sería el hacer la aplicación de preventa también de venta directa, añadiendo los módulos necesarios para ello como el de cobros o el de entrega.

Otra posible mejora, teniendo en cuenta que se están recogiendo las coordenadas de cada cliente visitado sería el pintar en un mapa la ruta realizada durante el día por los usuarios de la aplicación, de forma que se pudiesen llegar a optimizar.

Se podría estudiar el impacto en el rendimiento de la aplicación que tendría el envío de datos a través de canales más seguros, utilizando para ello mecanismos como el cifrado de datos.

También se podría mejorar el envío de las imágenes recogidas, ya que al tratarse como norma general de archivos de un peso considerable, se podría implementar un servicio para hacer el envío en segundo plano sin impactar sobre el uso de la aplicación.

Otro posible desarrollo que complementase este sería el desarrollo de un servidor de aplicaciones que además de ser capaz de recoger todos los datos enviados por los diferentes dispositivos, los tratase y administrase. Podría ofrecer un interfaz gráfico en el que administrar los diferentes datos e incorpóralos a una base de datos.

Capítulo VI. Glosario

- **AIDL**: siglas de Android Interface Definition Language, Lenguaje para la Definición de Interfaces en Android. Constituye un lenguaje de sintaxis muy básica que permite describir interfaces que pueden ser utilizadas de forma remota.
- **API**: siglas de Application Programming Interface, Interfaz de Programación de Aplicaciones. Consiste en un conjunto de llamadas que ofrecen acceso a funciones y procedimientos, representando una capa de abstracción para el desarrollador.
- **Breakpoint**: breakpoint o punto de interrupción. Término de programación, utilizado especialmente en la depuración de los programas. Se trata de una pausa intencionada y controlada durante la ejecución del programa.
- **Bytecode**: código intermedio, más abstracto que el código máquina, y que necesita de un mediador o máquina virtual para poder ser transformado y ejecutado en un hardware local.
- **Callback**: se denomina así a la relación que existe entre dos procesos cuando el origen de la comunicación es a su vez llamado o invocado por el proceso destino.
- **Checkbox**: elemento de interfaz de usuario que permite hacer una selección múltiple en un conjunto de opciones.
- **Cloud Computing**: computación en la nube concepto conocido también bajo los términos servicios en la nube, informática en la nube, nube de cómputo o nube de conceptos, del inglés *Cloud computing*, es un paradigma que permite ofrecer servicios de computación a través de Internet.
- **Dalvik**: nombre de la máquina virtual utilizada por el sistema operativo Android. Dalvik esta específicamente adaptada a las características de rendimiento de un dispositivo móvil y trabaja con ficheros de extensión ".dex", obtenidos desde el bytecode de Java.
- **EDGE**: siglas en inglés de Enhanced Data rates for GSM of Evolution, Tasas de Datos Mejoradas para la evolución de GSM. Es una tecnología para telefonía móvil que representa un puente entre la segunda y tercera generación de estos dispositivos.
- **Framework**: término con el que se define un amplio conjunto de elementos que permite desarrollar y organizar software utilizando un determinado lenguaje, sistema o tecnología. Habitualmente incluye bibliotecas, programas de desarrollo o manuales.
- **GPRS**: siglas de General Packet Radio Service, Servicio General de Paquetes vía Radio. es una extensión del estándar GSM que permite mejorar sus

prestaciones originales, como el envío de datos, uso de correo electrónico, navegación web o el aumento de las tasas de transferencia de datos.

- **GSM**: siglas de Groupe Spécial Mobile, más conocido como Sistema Global para las Comunicaciones Móviles, es el estándar más extendido para las comunicaciones con telefonía móvil. Permite llamadas, navegación por Internet o envío de SMS.
- **GUI**: siglas de Graphical User Interface, Interfaz Gráfica de Usuario. Representa la parte del software que, mediante un contexto o lenguaje principalmente visual y simbólico, permite al usuario utilizar una aplicación.
- **Hilo**: en sistemas operativos, un hilo constituye cada uno de los flujos de ejecución en el que puede ser divido un proceso. Todos los hilos de un proceso comparten espacio en memoria, archivos abiertos, variables globales, semáforos, etc. Permiten la ejecución concurrente de varias tareas. También llamado *thread*.
- **HTTP**: siglas de HyperText Transfer Protocol, Protocolo de Transferencia de Hipertexto. Constituye el protocolo utilizado para la transmisión de documentos a través de la Web entre un cliente y servidor.
- **Internet Engineering Task Force**: (IETF) (en español Grupo Especial sobre Ingeniería de Internet) es una organización internacional abierta de normalización, que tiene como objetivos el contribuir a la ingeniería de Internet, actuando en diversas áreas, como transporte, encaminamiento, seguridad.
- **JAR**: acrónimo de Java ARchive, en español Archivo Java. Representa una agrupación de varios ficheros Java y se usa generalmente para la distribución conjunta de clases y metadatos.
- **JVM**: siglas de Java Virtual Machine, en español Máquina Virtual de Java. Constituye un elemento software de la tecnología Java, encargado de transformar el código intermedio universal o bytecode en código máquina específico del hardware donde está instalado.
- **Kernel**: parte fundamental de un sistema operativo, responsable de facilitar acceso seguro al hardware, gestionar recursos y hacer llamadas al sistema. También conocido como núcleo.
- **Listener**: objeto que está a la espera de determinado evento.
- **Máquina virtual**: representa un software que emula el comportamiento de una determinada arquitectura o que permite adaptar un código fuente a las características de la máquina nativa.
- **Market de Android o Google Play:** Google Play Store o sólo Google Play (antes llamado Android Market) es una tienda de software en línea desarrollada por Google para los dispositivos Android. Es una aplicación que está preinstalada en la mayoría de los dispositivos Android y que permite a los usuarios buscar y descargar aplicaciones publicadas por desarrolladores terceros, alojada en Google Play.
- **MathML**: el MathML o *Mathematical Markup Language* es un lenguaje de marcado basado en XML, cuyo objetivo es expresar notación matemática de forma que distintas máquinas puedan entenderla, para su uso en combinación

con XHTML en páginas web, y para intercambio de información entre programas de tipo matemático en general.

- **Microkernel:** o mono núcleo es un tipo de núcleo de un sistema operativo que provee un conjunto de primitivas o llamadas al sistema mínimas, para implementar servicios básicos como espacios de direcciones, comunicación entre procesos y planificación básica.
- **MP3**: formato de compresión de audio digital cuyo nombre completo es MPEG-1 Audio Layer 3. También utilizado, por extensión, para nombrar al dispositivo móvil reproductor de audio digital en el que se almacena, organiza o reproduce archivos de audio digital.
- **Multitáctil:** es el nombre con el que se conoce a una técnica de interacción persona-computador y al hardware que la implementa. La tecnología multitáctil consiste en una pantalla táctil o touchpad que reconoce simultáneamente múltiples puntos de contacto, así como el software asociado a esta que permite interpretar dichas interacciones simultáneas.
- **Near Field Communication:** son las siglas en inglés de *Near Field Communication*, una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos a menos de 10cm.
- **Palmtop**: los dispositivos móviles (también conocidos como computadora de mano, *palmtop* o simplemente *handheld*) son aparatos de pequeño tamaño, con algunas capacidades de procesamiento, con conexión permanente o intermitente a una red, con memoria limitada, diseñados específicamente para una función, pero que pueden llevar a cabo otras funciones más generales.
- **PDA**: siglas de Personal Digital Assistant, en español Asistente Digital Personal. Dispositivo móvil utilizado como organizador personal, que cuenta generalmente con pantalla táctil, agenda, calendario, conectividad Wi-Fi, y aplicaciones ofimáticas, entre otros.
- **Plug-in**: componente de software que se relaciona y ejecuta con otro para aportarle una función nueva y generalmente muy especifica.
- **Proceso**: un proceso es un programa en ejecución, y representa la unidad de procesamiento básica gestionada por el sistema operativo.
- **Proxy:** en una red informática, es un programa o dispositivo que realiza una acción en representación de otro.
- **Push:** los sistemas Push o también llamados de recepción pasiva, son los sistemas en los que el servidor el que contacta con el cliente para informarle de la existencia de noticias.
- **RFC:** del inglés Request For Comment o Petición de Comentarios. Documentos que se iniciaron en 1967 que describen los protocolos de internet.
- **RMI**: siglas de Remote Method Invocation, en español Invocación de Métodos Remotos, es una tecnología de Java que permite comunicar objetos distribuidos escritos en este lenguaje.

- **Set-top box:** cuya traducción literal al español es caja que se coloca encima del televisor, es el nombre con el que se conoce el dispositivo encargado de la recepción y opcionalmente decodificación de señal de televisión analógica o digital (DTV), para luego ser mostrada en un dispositivo de televisión.
- **Sistema operativo**: programa cuya finalidad principal es simplificar el manejo y explotación de un elemento con capacidad computacional, gestionando sus recursos, ofreciendo servicios a las demás aplicaciones y ejecutando mandatos del usuario.
- **SDK**: siglas de Software Development Kit, en español Kit de Desarrollo de Software. Constituye un conjunto de herramientas que permiten a un desarrollador crear aplicaciones para una determinada plataforma o lenguaje.
- **SGML:** son las siglas de Standard Generalized Markup *Language* o Estándar de Lenguaje de Marcado Generalizado. Consiste en un sistema para la organización y etiquetado de documentos.
- **SmartPhone**: es un teléfono inteligente que puede comunicarse a través de Wi-Fi, bluetooth, conexión al Internet, envío de mensajería, emails. Generalmente se define como dispositivo electrónico de mano que integra la funcionalidad de un teléfono celular, PDA o similar. Generalmente se realiza añadiendo funciones de teléfono a un PDA existente o añadiendo funcionalidades "ınteligentes", como las funciones del PDA, en un teléfono celular. Una característica clave de un smartphone es que las aplicaciones adicionales pueden ser instaladas en el dispositivo. Las aplicaciones puede ser desarrolladas por el fabricante del dispositivo, por el operador o por cualquier empresa desarrolladora de software.
- **Software libre:** el software libre es una cuestión de la libertad de los usuarios de ejecutar, copiar, distribuir, estudiar, cambiar y mejorar el software. Más precisamente, significa que los usuarios de programas tienen las cuatro libertades esenciales. La libertad de ejecutar el programa, la libertad de estudiar cómo trabaja el programa y cambiarlo, la libertad de redistribuirlo y la libertad de distribuir copias de sus versiones modificadas.
- **Stylus**: navegación tipo Stylus es la que se basa en la utilización de un lápiz táctil.
- **SVG:** los Gráficos Vectoriales Escalables (del inglés *Scalable Vector Graphics*) o SVG es una especificación para describir gráficos vectoriales bidimensionales, tanto estáticos como animados (estos últimos con ayuda de SMIL), en formato XML.
- **Tackball**: un trackball es un dispositivo apuntador estacionario compuesto por una bola incrustada en un receptáculo que contiene sensores que detectan la rotación de la bola en dos ejes.
- **Timeout:** tiempo máximo que se debe esperar antes de abortar una tarea.
- **Touchscreen**: touchcreen o pantalla sensible al tacto utilizando rayos infrarrojos. Es un periférico de entrada/salida. Permite al usuario interactuar simplemente presionando sobre la opción que ve en la pantalla.

- **Trackpad**: el touchpad, trackpad, tapete táctil o alfombrilla táctil es un dispositivo táctil de entrada que permite controlar un cursor o facilitar la navegación a través de un menú o de cualquier interfaz gráfica.
- **UMTS**: siglas de Universal Mobile Telecommunications System, en español Sistema Universal de Telecomunicaciones Móviles. Constituye en estándar de comunicación para dispositivos de tercera generación o 3G, que ofrece capacidades multimedia y conexiones de alta velocidad en Internet.
- **URL**: siglas en inglés de Uniform Resource Locator, Localizador Uniforme de Recursos, es una secuencia de caracteres, de acuerdo a un formato modélico y estándar, que se usa para nombrar recursos en Internet para su localización o identificación, como por ejemplo documentos textuales, imágenes, vídeos, presentaciones digitales, etc.
- **Wi-Fi**: acrónimo de Wireless Fidelity, estándar de envío de datos que utiliza ondas de radio en lugar de cables.
- **WAP**: siglas de Wireless Application Protocol, en español o Protocolo de Aplicaciones Inalámbricas. Es un estándar para aplicaciones que utilizan las comunicaciones inalámbricas, como el acceso a servicios de Internet desde un teléfono móvil.
- **Widget**: componente gráfico utilizado en interfaces de usuario, con el cual el usuario puede interactuar, como por ejemplo cajas de texto, botones, ventanas, etc.
- **XHTML:** siglas de eXtensible HyperText Markup Language. XHTML es básicamente HTML expresado como XML válido. Es más estricto a nivel técnico, pero esto permite que posteriormente sea más fácil al hacer cambios o buscar errores entre otros.
- **XML**: siglas de Extensible Markup Language, en español Lenguaje de Marcado Extensible. Representa un lenguaje estándar que, mediante el uso de etiquetas y atributos, permite expresar e intercambiar fácilmente estructuras de datos.
- **World Wide Web Consortium:** el World Wide Web Consortium, abreviado W3C, es un consorcio internacional que produce recomendaciones para la World Wide Web.
- **3G**: es la abreviación de tercera generación de transmisión de voz y datos a través de telefonía móvil. La definición técnicamente correcta es UMTS (Universal Mobile Telecommunications System o Servicio Universal de Telecomunicaciones Móviles).

Capítulo VII. Bibliografía

Abc.es. (2011). *La evolución de los teléfonos móviles*. Recuperado el Marzo de 2012, de http://www.abc.es/20110322/tecnologia/abci-evolucion-telefonos-moviles-imagenes-201103221009.html

Android Developers. (2012). *Activating Components.* Recuperado el Abril de 2012, de http://developer.android.com/guide/topics/fundamentals.html#ActivatingComponents

Android Developers. (2012). *Application Fundamentals.* Recuperado el Abril de 2012, de http://developer.android.com/guide/topics/fundamentals.html#Components

Android Developers. (2012). *Broadcast Receiver.* Recuperado el Abril de 2012, de http://developer.android.com/reference/android/content/BroadcastReceiver.html

Android Developers. (2012). *Download the Android SDK.* Recuperado el Mayo de 2012, de http://developer.android.com/sdk/index.html

Android Developers. (2012). *Hardware options.* Recuperado el Mayo de 2012, de http://developer.android.com/guide/developing/devices/managing-avds.html#hardwareopts

Android Developers. (2012). *Intent.* Recuperado el Abril de 2012, de http://developer.android.com/reference/android/content/Intent.html

Android Developers. (2012). *Intent Resolution.* Recuperado el Abril de 2012, de http://developer.android.com/guide/topics/intents/intents-filters.html#ires

Android Developers. (2012). *Platform Versions.* Recuperado el Abril de 2012, de http://developer.android.com/resources/dashboard/platform-versions.html

Android Developers. (2012). *System Requirements.* Recuperado el Mayo de 2012, de http://developer.android.com/sdk/requirements.html

Android Developers. (2012). *The AndroidManifest.xml File.* Recuperado el Abril de 2012, de http://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Developers. (2012). *User Interface.* Recuperado el Abril de 2012, de http://developer.android.com/guide/topics/ui/index.html

Android Developers. (2012). *What is Android?* Recuperado el Marzo de 2012, de http://developer.android.com/guide/basics/what-is-android.html

Baz Alonso, A., Ferreira Artime, I., Álvarez Rodríguez, M., & García Baniello, R. (2009). *Dispositivos móviles.* Recuperado el Marzo de 2012, de http://156.35.151.9/~smi/5tm/09trabajos-sistemas/1/Memoria.pdf

Charte, F. (2001). *Psion y EPOC.* Recuperado el Marzo de 2012, de http://www.idg.es/pcworld/Psion_y_EPOC_Informatica_movil__IV_/art116340.htm

CNET. (2012). *http://news.cnet.com.* Recuperado el Abril de 2012, de http://news.cnet.com/8301-13579_3-20012418-37.html

Code.google.com. (2011). *SQLite Manager.* Recuperado el Mayo de 2012, de http://code.google.com/p/sqlite-manager/

Code.google.com. (2012). *Dalvik software*. Recuperado el Abril de 2012, de http://code.google.com/p/android-dalvik-vm-on-java/

Eclipse. (2012). *Eclipse Downloads.* Recuperado el Mayo de 2012, de http://www.eclipse.org/downloads/

Eclipse. (2012). *Eclipse Project*. Recuperado el 2012 de 2012, de http://www.eclipse.org/

Flosi, S. (2010). *comScore Reports September 2010 U.S. Mobile Subscriber Market Share.* Recuperado el Abril de 2012, de http://www.comscore.com/Press_Events/Press_Releases/2010/11/comScore_Reports_September_2010_U.S._Mobile_Subscriber_Market_Share

Gartner. (2012). *Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent Growth.* Recuperado el Mayo de 2012, de http://www.gartner.com/it/page.jsp?id=1924314

Google Play. (2012). *Barcode Scanner ZXing Team.* Recuperado el Mayo de 2012, de https://play.google.com/store/apps/details?id=com.google.zxing.client.android&feature=search_result#?t=W251bGwsMSwxLDEsImNvbS5nb29nbGUuenhpbmcuY2xpZW50LmFuZHJvaWQiXQ

Infojobs. (2012). *Evolución salarial de: programador.* Recuperado el Mayo de 2012, de http://salarios.infojobs.net/resultados.cfm?sueldo=programador

Java.com. (2012). *Conozca más sobre la tecnología Java*. Recuperado el Mayo de 2012, de http://www.java.com/es/about/

Linux Para Todos. (2012). *Sistema Operativo.* Recuperado el Marzo de 2012, de http://www.linuxparatodos.net/portal/staticpages/index.php?page=sistema-operativo

Microsoft. (2012). *ASP.NET.* Recuperado el Junio de 2012, de http://www.asp.net/

Parrés, I. (2011). *Los dispositivos móviles, el futuro de las empresas .* Recuperado el Marzo de 2012, de http://www.rrhhpress.com/index.php?option=com_content&view=article&id=11398:los-dispositivos-moviles-el-futuro-de-las-empresas&catid=81:europe&Itemid=197

Slobojan, R. (2007). *Dalvik, Android's virtual machine, generates significant debate.* Recuperado el Mayo de 2012, de http://www.infoq.com/news/2007/11/dalvik

Smart-gsm.com. (2011). *Características técnicas Samsung Galaxy Ace S5830.* Recuperado el Mayo de 2012, de http://www.smart-gsm.com/moviles/samsung-galaxy-ace-s5830

Sony. (2012). *VAIO Serie F.* Recuperado el Mayo de 2012, de http://www.sony.es/product/vn-f-series

Sqlite.org. (2012). *Sqlite*. Recuperado el Mayo de 2012, de http://www.sqlite.org/

Steven, T. (2010). *Google expands Android's reach.* Recuperado el Abril de 2012, de http://www.engadget.com/2010/10/01/google-expands-androidss-reach-accepting-paid-apps-from-20-mor/

TICbeat. (2011). *Ya se activan más de 700.000 Android cada día.* Recuperado el Abril de 2012, de http://www.ticbeat.com/sim/activan-700000-android-dia/

W3C. (2012). *HTTP - Hypertext Transfer Protocol*. Recuperado el Mayo de 2012, de http://www.w3.org/Protocols/

W3C. (2012). *XML*. Recuperado el Mayo de 2012, de http://www.w3.org/XML/

Wikipedia. (2012). *BlackBerry OS*. Recuperado el Marzo de 2012, de http://es.wikipedia.org/wiki/BlackBerry_OS

Wikipedia. (2012). *Dalvik (software).* Recuperado el Abril de 2012, de http://en.wikipedia.org/wiki/Dalvik_%28software%29

Wikipedia. (2012). *Palm OS.* Recuperado el Marzo de 2012, de http://es.wikipedia.org/wiki/Palm_OS

Windows. (2012). *IIS* . Recuperado el Junio de 2012, de http://www.iis.net

Zxing. (2012). *Multi-format 1D/2D barcode image processing library with clients for Android.* Recuperado el Mayo de 2012, de http://code.google.com/p/zxing

Anexo. Servidor de aplicaciones

En este anexo se va a hacer referencia al servidor de aplicaciones que se ha implementado para poder llevar a cabo las pruebas de comunicaciones de la aplicación. El servidor de aplicaciones se ha desplegado en un servidor IIS (*Internet Information Server*) y va a tener como finalidad el permitir hacer el intercambio de ficheros con los dispositivos.

A continuación se va a hacer una referencia a la pagina *.asp* que formaría el servidor y contra la cual la aplicación desarrollada en este proyecto realiza las comunicaciones. El único requerimiento que debe tener es tener publicada una interfaz preparada para publicar los ficheros que se van a descargar y recibir los ficheros que el dispositivo envía. En este caso se ha utilizado la siguiente pagina asp.

Esta página *.asp* tiene un funcionamiento sencillo y se va a encargar de recepcionar los ficheros enviados por los dispositivos, y por ejemplo en caso de que ya hubiese un fichero enviado por el dispositivo móvil, al volver a comunicar se encargará de concatenarlo con el ya existente para que no se produzcan perdidas de datos.

```
Sub RecibeFicheroDiario
	Dim fs, f, fq , ts, s,cadena,i , varext,varbr , fD , fO
	Dim total
	Dim leidos
	Dim obj
	Dim totalbytes
	Dim ahora , scad
	Dim fichero , ficheroO,ficheroD
	Set fs = CreateObject("Scripting.FileSystemObject")
		if err then
			call errores ("511")
			exit sub
		end if
		If vartxt = "1" Then
			varext=".log"
			varbr=" "
		else
			varext=".html"
			varbr="<br>"
		end if
		fichero = Server.MapPath( exlog & "LOG" & DatePart("yyyy",date) &
DatePart("m",date)& DatePart("d",date) & varext )

		if err then
			call errores ("521")
			Exit sub
		end if
		Set fq = fs.OpenTextFile( fichero , 8 , True )
		if err then
```

```
                  call errores ("522")
                  exit sub
            end if
            fq.write "COMIENZO DE RECEPCION DE DATOS: " & exin & varbr &
vbcrlf
            fq.Close
            fichero = Server.MapPath( path & Request.QueryString( "palmtop" )
& "/in/" & "aux" & exin  )
            if err then
                  call errores ("521")
                  exit sub
            end if
            Set f = fs.OpenTextFile( fichero , 2 , True )
            'Guardamos en una variable date de inicio de comunicacion
            ahora=now
             f.write "[ini TX: " & ahora &"]" & vbcrl
            if err then
                  call errores ("522")
                  exit sub
            end if
            total = 0
            totalbytes = Request.TotalBytes
            Do While total < totalbytes
                  leidos = 1024
                  cadena=Request.BinaryRead(leidos)
                        for i=1 to LenB(cadena)
                              f.write Chr(AscB(MidB(cadena,i,1)))
                        next
                        total = total + leidos
            Loop
            ahora=now
            f.write  "[end TX: " & ahora &"]"& vbcrlf
            f.Close
            ficheroO = Server.MapPath( path & Request.QueryString( "palmtop"
) & "/in/" & "aux" & exin  )
            ficheroD = Server.MapPath( path & Request.QueryString( "palmtop"
) & "/in/" & exin  )
            Set fD = fs.OpenTextFile( ficheroD , 8 , True )

            Set f = fs.GetFile( ficheroO )
            Set ts = f.OpenAsTextStream(1, -2)
            Do While Not ts.AtEndOfStream
                  scad = ts.ReadLine
                  fD.WriteLine scad
            Loop
            ts.Close
            fD.Close
            fs.DeleteFile ficheroO , True
          call crearusr
            fichero = Server.MapPath( exlog & "LOG" & DatePart("yyyy",date) &
DatePart("m",date)& DatePart("d",date) & varext )
            if err then
                  call errores ("521")
```

```
                Exit sub
            end if
            Set fq = fs.OpenTextFile( fichero , 8 , True )
            if err then
                call errores ("522")
                exit sub
            end if
            fq.write "FIN DE RECEPCION DE DATOS" & varbr & vbcrlf
             fq.Close
            Set f = Nothing
            Set fs = Nothing
            Set obj = Nothing
            Set fD = Nothing
            Set ts = Nothing
      End Sub
```